João Pedro Botequim Ferreira

Licenciado em Engenharia Informática

Mobile Image Processing to Support Educational Geometry Games

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Nuno Manuel Robalo Correia, Prof. Catedrático, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Júri:

Presidente:	Doutor Pedro Manuel Corrêa Calvente Barahona
Arguente:	Doutora Maria Teresa Caeiro Chambel
Vogal:	Doutor Nuno Manuel Robalo Correia

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2014

Mobile Image Processing to Support Educational Geometry Games

To my parents and my cat

Acknowledgements

Os agradecimentos. Apesar de haver total liberdade no conteúdo e forma desta secção, normalmente inicia-se com os agradecimentos institucionais (orientador, instituição, bolsas, colegas de trabalho, ...) e só depois os pessoais (amigos, família, ...)

Abstract

The mobile IT era is here, it is still growing and expanding at a steady rate and, most of all, it is entertaining. Mobile devices are used for entertainment, whether social through the so-called social networks, or private through web browsing, video watching or gaming. Youngsters make heavy use of these devices, and even small children show impressive adaptability and skill. However not much attention is directed towards education, especially in the case of young children. Too much time is usually spent in games which only purpose is to keep children entertained, time that could be put to better use such as developing elementary geometric notions.

Taking advantage of this pocket computer scenario, it is proposed an application geared towards small children in the 6 – 9 age group that allows them to consolidate knowledge regarding geometric shapes, forming a stepping stone that leads to some fundamental mathematical knowledge to be exercised later on.

To achieve this goal, the application will detect simple geometric shapes like squares, circles and triangles using the device's camera. The novelty of this application will be a core real-time detection system designed and developed from the ground up for mobile devices, taking into account their characteristic limitations such as reduced processing power, memory and battery. User feedback was be gathered, aggregated and studied to assess the educational factor of the application.

Keywords: application, prototype, detection, geometric shapes, geometric notins, camera, mobile

Resumo

A era móvel está aqui, está ainda a crescer e a expandir a um ritmo constante e, acima de tudo, é entretedora. Os dispositivos móveis são usados primariamente para entretenimento, seja social através das chamadas redes sociais, ou privado através de navegação na internet, visualização de vídeos ou jogando videojogos. Os mais jovens fazem uso elevado deste dispositivos, e mesmo crianças pequenas demonstram uma adaptabilidade e destreza impressionantes. No entanto, pouca atenção é direcionada para a educação, especialmente no caso de crianças pequenas. Demasiado tempo é normalmente gasto em jogos fúteis cujo único propósito é manter crianças entretidas, tempo esse do qual poderia ser feito um uso melhor tal como o desenvolvimento de noções geométricas elementares.

Aproveitando este cenário do computador de bolso, é proposta uma aplicação direcionada para crianças pequenas na faixa etária dos 6 aos 9 anos que lhes permita consolidar conhecimento sobre formas geométricas, formando uma base que levará a um conhecimento matemático fundamental que será exercitado no futuro. É um videojogo, uma vez que os videojogos são o principal chamariz de atenção para pessoas nesta faixa etária. A ideia é seguir uma aproximação de exemplos no mundo real e fazer com que as crianças resolvam questionários geométricos simples utilizando a câmera do dispositivo, eventualmente levando a noções mais complexas como paralelismo de linhas e simetria de figuras.

Para atingir este objectivo, a aplicação irá detetar formas geométricas simples como quadrados, círculos e triângulos utilizando a câmera do dispositivo, levando em conta as limitações características móveis tais como poder de processamento, memória e bateria reduzidos. Feedback dos utilizadores irá ser coletado, agregado e estudado afim de avaliar o factor educacional da aplicação.

Palavras-chave: jogo, deteção de formas, formas geométricas, câmera, móvel

Contents

List of Figures

List of Tables

1

Introduction

It was not long ago that mobile devices were introduced to the masses, and the smartphone, tablet and *phablet* boom is still well underway, showing no signs of slowing down (Telegraph, 2013). With each iteration these devices become faster, lighter, thinner, last longer and this fosters the need to upgrade to a new device for various reasons. In the case of most families with small children, devices are firstly shared and soon become inherited with the acquisition of newer, more advanced successors. In other cases, it is the young who crave for this technology and usually are better versed than the parents themselves, with infants showing remarkable adaptability, to the point of discarding other types of media as non-functional (News, 2011). Therefore there is a considerable number of children either owning or with easy access to mobile devices and using them for all kinds of activities, mostly related to leisure and entertainment.

The idea is to take advantage of this scenario and bestow more educational content upon the children while they are using a smartphone or tablet, by means of an application which will allow them to consolidate knowledge about geometric shapes and notions.

1.1 Motivation

As previously stated, young children see mobile devices as mobile entertainment and tend to engage on less educational content while using them. Mobile games tend to be the preferred choice and with readily available application distribution platforms such as App Store [1] and Google Play [2] which sport thousands of low-cost and free mobile games. There is seemingly no end to this content nor reasons to engage on something else, at least while the battery lasts.

Albeit not new, the idea of using games for educational purposes seems to be gathering strength due to the new possibilities mobile devices brought to the realm of interactivity. One area susceptible to children within the 6 – 10 age group is the contact with geometric shapes, which plays an important role in developing recognition skills that are later used not only in mathematical, geological, drawing and other fields, but throughout their lives in day to day activities like driving, even if at a more subconscious level. A good, solid knowledge base is thus fundamental. Since games are the primary attention grabber for this age group, one would expect to encounter a considerable amount of education-oriented games in the aforementioned distribution platforms, and indeed there are some that deal with geometry, however most are preset quizzes with fixed answers that don't really take advantage of the mobile device's array of capabilities. Also there is the common notion that humans tend to learn better by example, especially if one uses real world examples.

This brings motivation to create a simple yet novel geometry game where answers are not preset, but can be many things found in the real world, and further expand the interactive learning area. That said, creating a full-fledged game with a real-time detection system falls beyond the scope of time available for this dissertation, moreover when it has to be build from the ground up especially for mobile devices. Thus, only the core detection system will be developed.

[1] http://www.apple.com/osx/apps/app-store.html
[2] https://play.google.com/store

1.2 Context

The present dissertation emerges as a collaboration effort between the startup company Watizeet [3], the initial proposer, and CITI [4] (Center for Informatics and Information Technologies) located in the Department of Informatics of the Faculty of Sciences and Technology of the New Lisbon University. The dominant areas of this work are image processing, multimedia and gaming.

1.3 Expected Contributions

A number of contributions is to be expected from this dissertation, a proof-of-concept Android [5] application prototype capable of detecting simple geometric shapes and notions such as line parallelism, angles, and symmetry. Further on a study will take place to assess the educational factor of the application.

Mobile-oriented library modules To the author's knowledge, at the time of this writing there are no libraries or set of functions that will accomplish what is described above efficiently in the given context. While OpenCV is adequate for this sort of tasks, most detection code written in OpenCV does not take into consideration the generally low processing power of mobile devices. Therefore a library with a new set of mobile-oriented geometry detection modules is another expected contribution.

Android application prototype While the longterm goal is to reach as many users as possible, as of 2013 Android is the leading mobile platform, dominating over 80% of the market (CNET, 2013), making it the primary development platform. The application will also make use of OpenCV [6], a highly popular open source library of programming functions mainly aimed at real-time computer vision. It is expected that the detection system at the core of this prototype will be integrated in an actual mobile game to be developed by Watizeet.

Assessment on the educational factor In order to assess the educational factor, a study will be undertaken together with the target audience of the application and any feedback will be taken into account for further improvements.

[3] http://watizeet.com/
[4] http://citi.di.fct.unl.pt/
[5] http://www.android.com/
[6] http://opencv.org/

1.4 Document Structure

The document is divided into seven chapters: introduction, related work, mobile-oriented library, system prototype, evaluation, user study and finally conclusion and future work.

Introduction Chapter 1 presents an overview of the dissertation regarding its description, context and expected contributions.

Related Work Chapter 2 sums what other work has been done in two main related areas – **Shape Recognition** and **Virtual and Augmented Reality in Geometry Education**.

Mobile-Oriented Library Chapter 3 details the requirements, module structure and functions of the mobile-oriented library.

System Prototype Chapter 4 explains in greater detail the development of the Android system prototype. It provides insight into the architecture, integration with the mobile-oriented library, user interface and detection modes.

Evaluation Chapter 5 analyses the results obtained from tests done to the prototype integrating the mobile-oriented library on different mobile devices and configurations.

User Study Chapter 6 reports the results gathered from an experiment conducted in an elementary school in order to access the educational factor of the application.

Conclusion and Future Work Chapter 7 critiques and comments on the work carried out during the course of this dissertation, along with possible improvements outlined as future work.

2

Related Work

This chapter presents a study of work done in the areas relevant to the present dissertation. It is divided into two main sections, the first (Section 2.1) dedicated to shape recognition and the latter (Section 2.2) to virtual reality in geometry learning.

2.1 Shape Recognition

Although object and shape recognition has been around for quite some time, it was only recently that it gained a broader research activity (Szeliski, 2011). Machine vision was mostly confined to the industrial segment, applied to conveyor belt inspecting, metal cutting and other well defined, immutable, mass-producing tasks. With the development of better equipments, particularly all-in-one devices such as smartphones and tablets, there was a sudden urge to bring object recognition into the street and use it to, among other things, explore and enhance the dynamic, ever-changing world around us. If a picture is worth a thousand words, a recognized one is possibly worth more.

In order to do any kind of object recognition, a few steps must be taken. This section goes over the topics of feature extraction and matching, showing some of the most relevant methods and techniques in the context of mobile devices, finishing with an example of a full-fledged shape recognition system.

2.1.1 Feature Extraction

Feature extraction is the process of transforming input data into a reduced set of features in order to perform some desired task using this reduced representation as opposed of the full size input. This is broadly used as a starting point for several computer vision and image processing algorithms. To better understand feature extraction, one must begin by understanding what a "feature" is. Indeed a feature can be many things, and while there is no canon definition of what constitutes a feature, it can be defined as an "interesting" part of an image (Szeliski, 2011), or as a way to describe some object using reduced (yet sufficiently accurate for the task at hand) notions. For instance, a square can be defined as a group of four vertices separated by equal distances, instead of a full fledged group of four lines intersecting each other and forming 90 degree angles at the endpoints; whereas a circle can be defined by its compactness index instead of a given centre point and a radius. Features are used not only as a starting point but also as main primitives for several algorithms, therefore any given algorithm will often be as good as its feature extractor. This is all the more important in the mobile devices' realm, where processing power, memory, storage and operational time is limited.

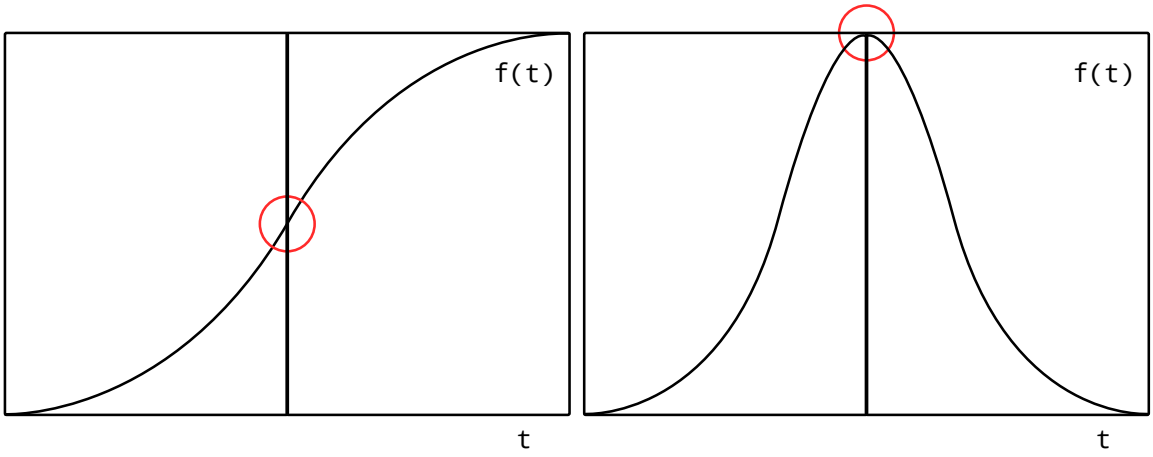
Taking these limitations into consideration, a feature extractor must be light, fast and relatively accurate in order to prove viable in such context. Below are presented a number of features and their extraction methods and techniques that fit this criteria.

2.1.1.1 Edges

Edge detection is the process of identifying points in an image where there are abrupt changes in its brightness, also referred to as intensity. The aim is to obtain a set of connected curves that represent the boundaries of objects, as well as curves that correspond to discontinuities in surface orientation. The resulting image will have filtered out information that is supposedly regarded as less relevant, while preserving the structure of the objects. Therefore any subsequent task of interpreting the information contents may be considerably simplified comparing to using the original image. Several edge detection methods have been proposed, such as the Sobel and Canny operators.

Sobel The Sobel operator is based on the fact that the pixels show a high intensity variation in edge areas. Calculating the first derivative of the intensity, it is observed the edge is characterized by a maximum, as shown in Figure 2.1. It can be seen as a measure of the image variation in the vertical

and horizontal directions, mathematically called a gradient and defined as a 2D vector composed of the function's first derivatives (Equation 2.1). Usually a 3×3 kernel matrix, or pixel window, is used to operate on the pixel of interest in order to reduce influence of noise. Figure 2.2 shows the result of an application of the Sobel operator.

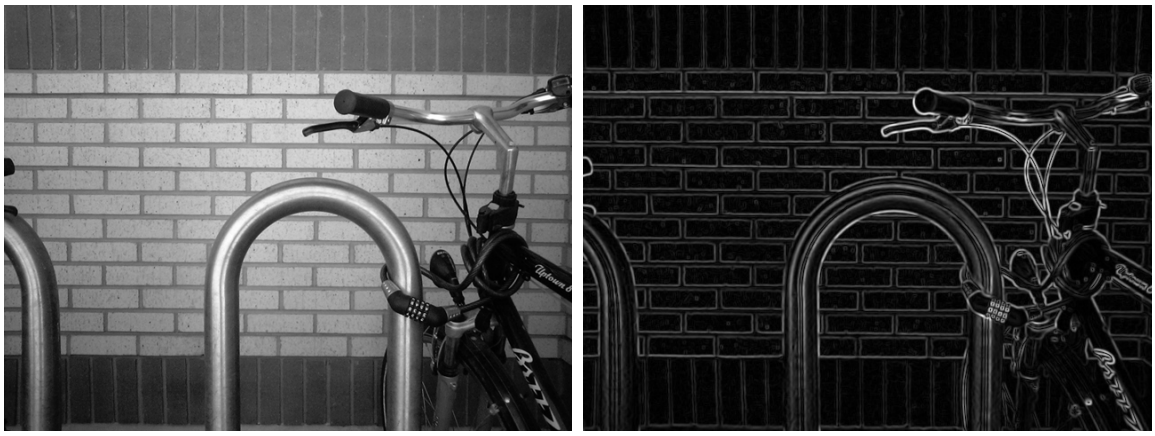


(a) The intensity function $f(t)$ alongside an edge

(b) The first derivative of $f(t)$. The maximum indicates the presence of an edge

Figure 2.1: Graphical example of a high intensity variation

$$grad(I) = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]^{\mathbf{T}} \tag{2.1}$$



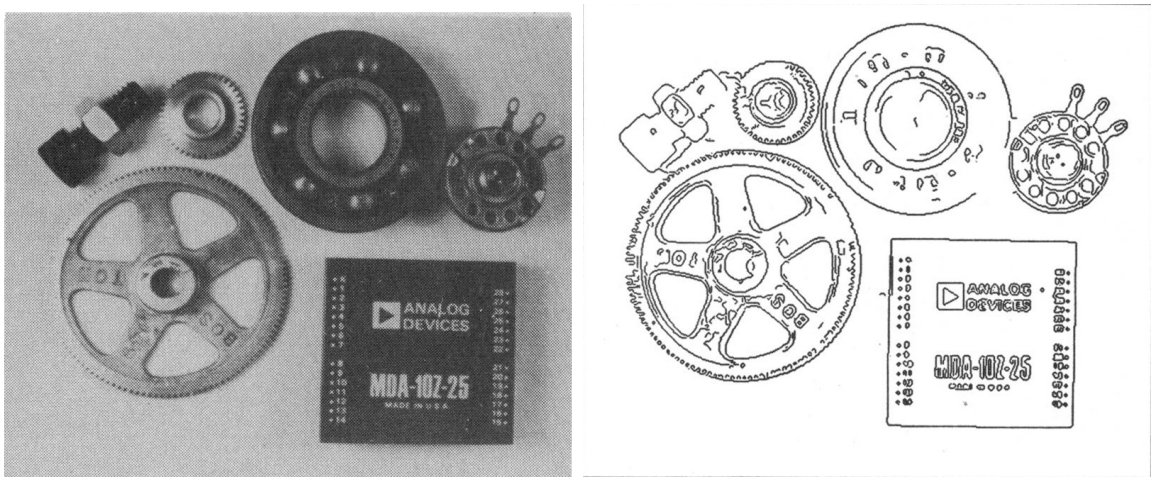
(a) Original image

(b) Edges image

Figure 2.2: An example of Sobel edge detection[1]

[1]Taken from http://en.wikipedia.org/wiki/Sobel_operator

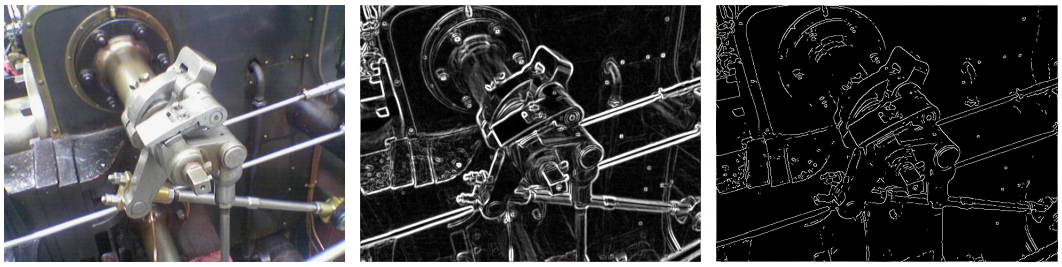
Canny The Canny operator, developed by Canny (1986), is a widely used multi-stage edge detector algorithm that strived for good detection, localization and minimal response. Good detection and localization means it would identify as many correct edges as possible in the image, and such edges would be marked as close as possible to the real edge, respectively. Minimal response would be the ability to mark any given edge only once while avoiding identifying false edges due to image noise. The are four stages to this algorithm. The first is to smooth out the image using a Gaussian filter, since the edge detection in general is susceptible to noise (Szeliski, 2011). Next, the smoothened image is filtered with a kernel in both horizontal and vertical direction in order to get the respective first derivatives to ultimately obtain the edge gradient, which is always perpendicular to edges, and direction for each pixel. Afterwards, a full scan of the image is carried out to remove any extraneous pixels which may not be a part of the edge itself, resulting in a binary image with "thin" edges. Finally the last stage uses hysteresis thresholding to identify genuine edges among the set of all edges, discarding noise pixels that are sure not to belong to any genuine edge. Figure 2.3 shows the result of an application of the Canny operator.



(a) Original image (b) Edges image

Figure 2.3: An example of Canny edge detection (Canny, 1986)

Out of the two operators, the Canny operator, while not the fastest, yields better edge detection results, which can induce performance gains in later stages of a shape recognition algorithm. Figure 2.4 shows a comparison of the Sobel and Canny operators applied to the same image.



(a) Original image (b) Sobel image (c) Canny image

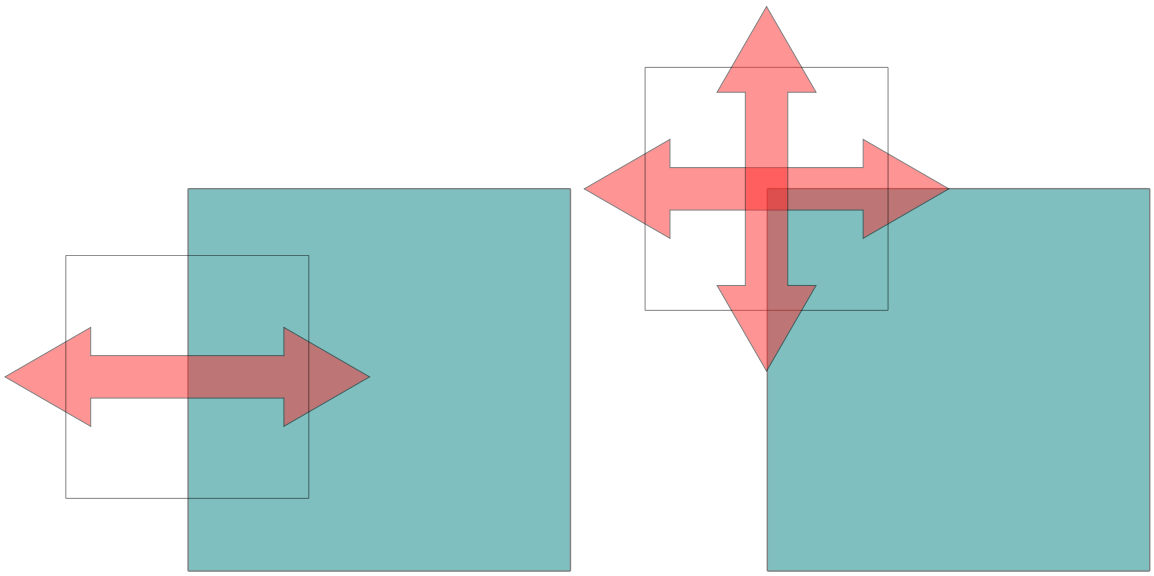
Figure 2.4: Comparing the Sobel and Canny operators

2.1.1.2 Corners

While corner detection is pretty self-explanatory, most corner detection methods detect feature points, rather than corners in particular. A feature point, also known as keypoint or interesting point, is a point in an image which is usually well defined and can be consistently detected (Szeliski, 2011). Corners stand out as one such feature point, and can be defined as the intersection of two edges, or as a point for which there are two dominant and different edge directions regarding the surrounding area. They are also plentiful in images containing man-made objects, which could prove advantageous when detecting simple shapes through their vertices. Some relevant, corner-specific detection methods are introduced next.

Harris The Harris corner detection algorithm (Harris and Stephens, 1988), while also being able to detect edges, is primarily used for identifying corners in an image. It is based on the idea that any given region in the image has a corner in it if the pixels in said region cover two dominant, orthogonal gradient directions. This is illustrated in Figure 2.5. In essence, the Harris corner detection algorithm scans the image regions, and, for each region, analyzes the eigenvalues of the covariance matrix of its pixels' gradient vectors. The eigenvectors of a covariance matrix point along prominent directions where points are found, and the corresponding eigenvalue magnitude indicates how prominent the direction is. A magnitude threshold is chosen, based on which the presence of corners is verified through the number of eigenvalues whose magnitude is higher than the threshold. If no eigenvalues are above the threshold, it is most likely a flat surface; if a single eigenvalue is above the threshold, then it is regarded as an edge, for it has a single dominant direction; if two eigenvalues are above the threshold, it means there are two dominant, orthogonal directions – a corner as defined above. Figure 2.6 shows the result of an application of the Harris corner

detection algorithm.



(a) A single dominant gradient direction indicates an edge

(b) Two dominant gradient directions indicate a corner

Figure 2.5: The idea behind Harris corner detection

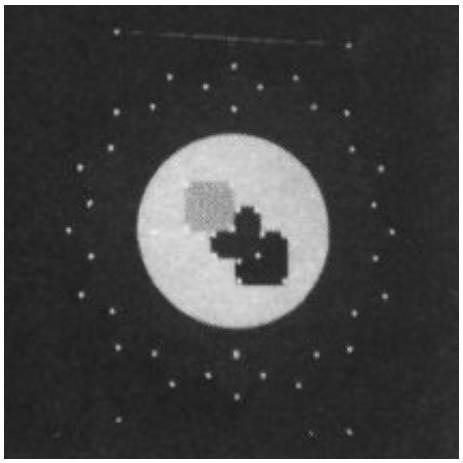


Figure 2.6: An example of Harris corner detection (Harris and Stephens, 1988)

FAST Features from Accelerated Segment Test (FAST) is a high-speed corner detection method proposed by Rosten and Drummond (2006) which aimed to uphold its namesake, and is targeted at real-time frame-rate applications. Its novelty lies in using machine learning algorithms to yield a large speed increase without necessarily sacrificing accuracy. Like Harris, FAST is also tied to the definition it derives of what constitutes a corner. However, this definition is now based on the intensity of the pixels around a candidate corner, and the decision to accept or reject the candidate as such is carried

out by examining a circle of pixels, denominated a Bresenham circle, centered on it. Figure 2.7 exemplifies this notion. Should an arc, composed by $3/4$ of the circle's perimeter pixels, be found in which every pixel's intensity differs in module by a certain threshold from the centre's intensity, then a corner is declared. This allows for another speed improvement. Instead of testing the circle pixel by pixel, one can test four pixels evenly separated on the circle, *i.e.*, top, right, bottom and left pixels, it can be shown (Rosten and Drummond, 2006) that at least three of these pixels' intensities must all differ from the centre pixel. If it is not the case, then the candidate pixel can be immediately rejected without examining additional perimeter pixels. In practical means, this is very effective since most of the candidate corners will be rejected by this simple comparison test.

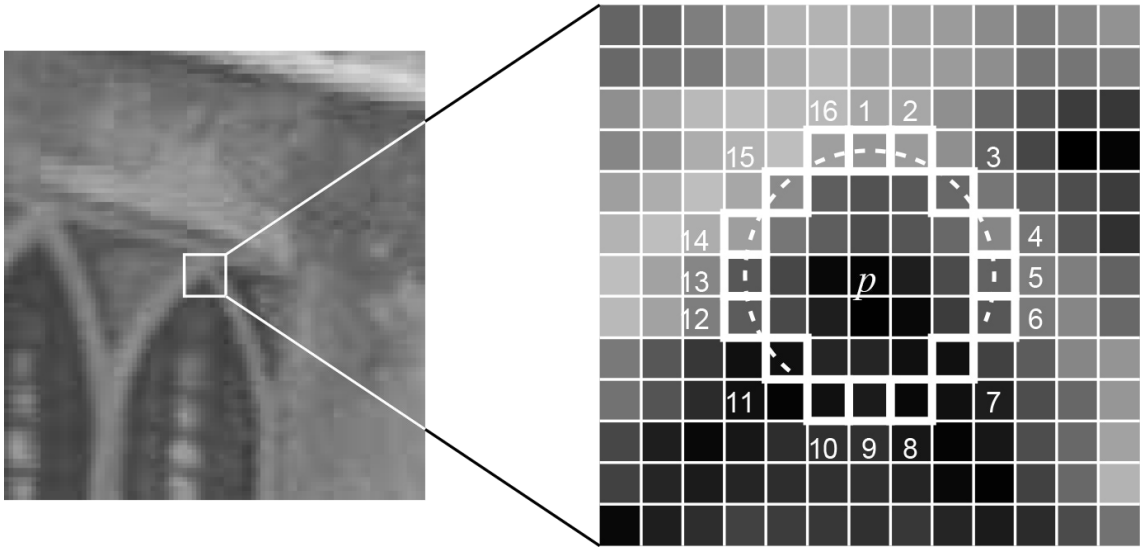


Figure 2.7: A Bresenham circle being used in a candidate corner test. The highlighted squares are the pixels comprising the circle perimeter and the pixel p at the centre is the candidate corner. The dashed arc marks the 12 ($3/4$ of 16) contiguous pixels which have a higher intensity comparing to p than the threshold (Rosten and Drummond, 2006)

While the Harris corner detector is widely used it is not adequate for real-time processing as it cannot operate at full frame rate (Rosten and Drummond, 2006), whereas FAST not only is several times faster, but can outperform other corner detectors and operate on low power hardware according to Rosten and Drummond (2006). The latter claim is indeed relevant in the context of this dissertation, and will actively be looked into.

2.1.2 Matching

After extracting the relevant features for a certain task, the next step is to match the features from two or more images in order to identify some image, whether for labeling and storage purposes, or for further downstream processing. There are plenty of ways to match features, with varying degrees of precision, ranging from pure brute force to statistical matching and decision making. However it is clear that several methods are not viable where mobile devices are concerned. In the case of feature extraction (Section 2.1.1), feature matching methods must be computationally efficient while preserving some degree of accuracy.

2.1.2.1 Edge Pixel Point Eigenvalues

This algorithm, developed by Chen et al. (2010), uses the eigenvalue of pixel points located on the edges to quickly recognize polygon apexes, or vertices, and rank order. It is based on the following definitions:

- The distance along the polygon edges of a pixel point P and its eigenvalue follow-pixel point P', called the eigenedge-distance of P, d_{EED}.
- The straight-line distance between P and P', called the eigendistance of P, d_{ED}, is fixed and depends on the size of the recognized figure.
- The eigenvalue of P, v_{EV}, which is given by Equation 2.2.

$$v_{EV} = \big||x - x'| - |y - y'|\big| \tag{2.2}$$

Figure 2.8 exemplifies the aforementioned definitions. Note that P's d_{EED} and d_{ED} are equal in Figure 2.8.

The underlying principle details that, if any two given pixel points P and Q and their respective eigenvalue follow-pixel points P' and Q' are situated on the same edge, the v_{EV} of P and Q is equal, if and only if the edge is not a curve. On the other hand, if a pixel point P and its eigenvalue follow-pixel point P' are not situated on the same edge, the v_{EV} of P varies continuously in the same search direction movement, either clock or anti-clockwise. Knowing this, Chen et al. (2010) state that the number of apexes of a polygon are equal to the number of times that the v_{EV} of all pixel points on the polygon edges change from steady to variation along a search direction. This can also be applied to recognize circles and ellipses. Given a fixed d_{EED}, the d_{ED} of each pixel point along the circumference is the same in relation to each other for the first, but different for the latter.

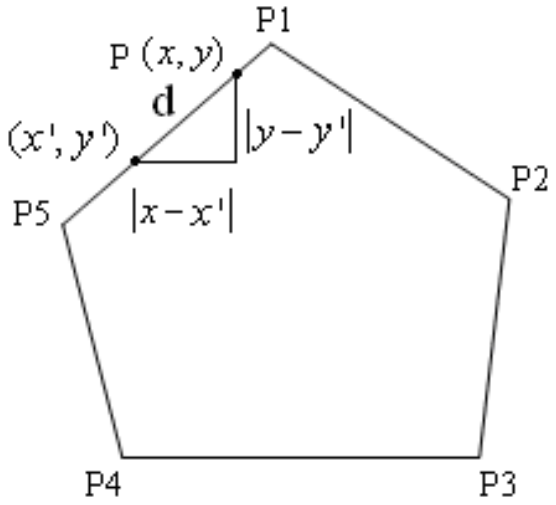


Figure 2.8: An example of the definitions (Chen et al., 2010)

The midpoint between the pixel point R with minimum d_{ED} and its eigenvalue follow-pixel point R' along the edge is the spot where the ellipse and its semi-major-axis cross. Likewise, a midpoint with maximum d_{ED} is where the ellipse and its semi-minor-axis cross. According to Chen et al. (2010), many advantages stand out from this algorithm, with the most relevant for this dissertation being summarized below:

1. For a figure resolution size of $W * H$, a time and space complexity of $O(n)$ can be achieved, with $n(n << W * H)$ as the number of edge pixel points.
2. A lower time and space complexity than other Hough transform-based recognition algorithms, while also having a lower implementation difficulty.
3. No need for shape descriptors or previous template matching.
4. Invariant to scale, translation and rotation.

This algorithm can be applied to recognize a myriad of simple shapes such as triangles, squares, pentagons, hexagons in addition to circles and ellipses as mentioned previously.

2.1.2.2 Compactness

Another way to recognize simple shapes is through the shape's own compactness (Pomplun, 2013). A shape compactness is nothing more than its perimeter squared divided by its area as shown in Equation 2.3.

$$c = \frac{P^2}{A} \tag{2.3}$$

This computation is applicable to all geometric shapes, independent of scale and orientation, and its value is dimensionless (Zakaria, Choon, and Suandi, 2012):

- A square of height s has a perimeter $P = 4s$ and an area $A = s^2$, therefore its compactness $c = \frac{16s^2}{s^2} = 16$.
- A circle of radius r has a perimeter $P = 2\pi r$ and an area $A = \pi r^2$, therefore its compactness $c = \frac{4\pi^2 r^2}{\pi r^2} = 4\pi$.

The more compact a shape is, the lower its compactness value, with no shape being more compact than a circle. Thus 4π is the minimum value for compactness (Pomplun, 2013). This simple calculation is not only considerably fast and efficient computationally, but it is also highly accurate when applied to simple shapes as demonstrated by Zakaria, Choon, and Suandi (2012). Thus it is a viable method for shape recognition on mobile devices.

2.1.2.3 Discussion

Overall the two methods are adequate for deployment in a mobile device application, each with its own reasons. The Edges Pixel Point Eigenvalues (Section 2.1.2.1) method can not only recognize the same shapes as the Compactness (Section 2.1.2.2) method, as it can also recognize shapes with different side lengths *i.e.*, rectangles, non-equilateral triangles, trapezoids, etc., and even non-convex shapes, which are beyond the Compactness method since they do not possess defined perimeter and area formulae. Despite that, the Compactness method excels at recognizing equilateral shapes and circles, and has been deployed to some extent in shape recognition application for mobile devices (Kareva, 2011a). Perhaps a combination of both methods will harness the best of both worlds.

2.1.3 A Shape Recognition System

In order to gain a better understanding of how shape recognition is carried out, one must look at a full-fledged shape recognition system (Zakaria, Choon, and Suandi, 2012). It is divided in multiple stages, as depicted in Figure 2.9, taking as input an RGB image and going through all the stages to produce an output image with the target shapes segmented.

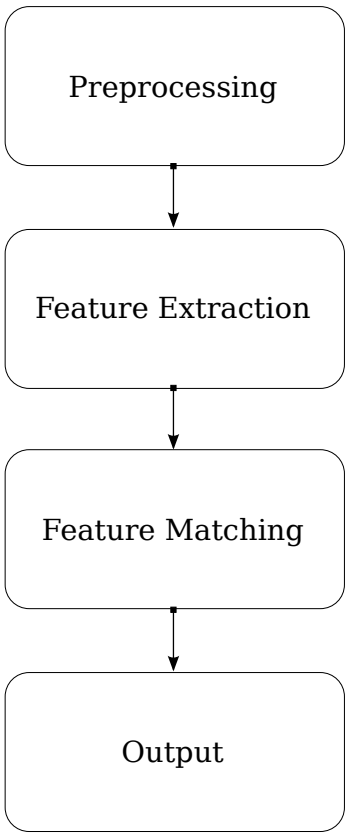
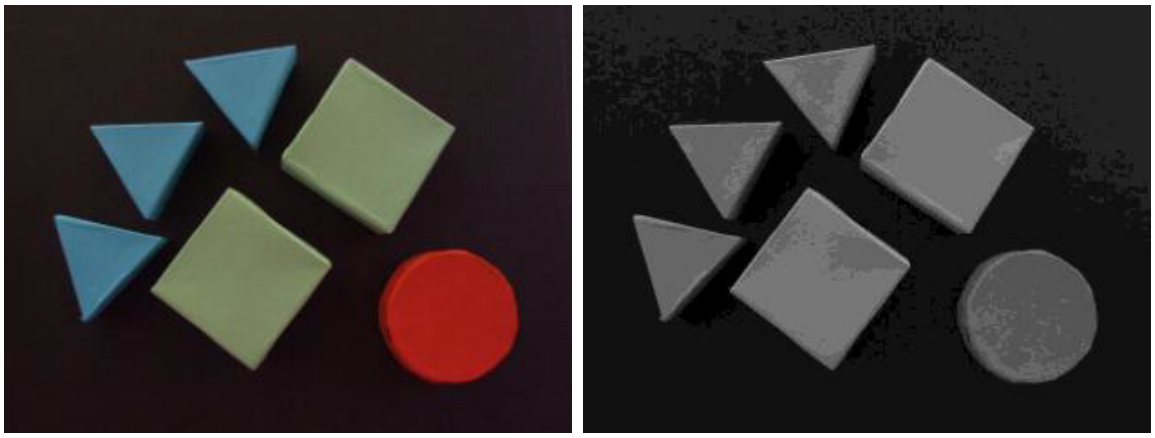


Figure 2.9: Block diagram of the multi-stage system

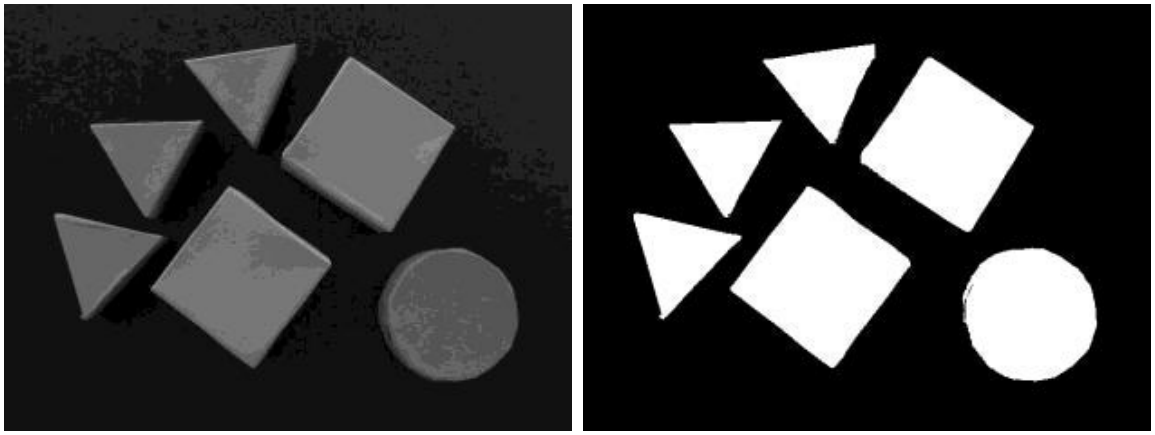
Preprocessing The first stage of the system is where the input image is preprocessed with the aim of helping the work done by later stages which allows them to achieve better results, while also potentially reducing their workload. In this stage, Zakaria, Choon, and Suandi (2012) start by changing the image's color space from RGB to HSL in order to work with the L (Lightness) channel. By using the L channel, a good color separation between the objects and the background can be achieved (Figure 2.10). It is then applied an effective thresholding technique developed by Otsu (1979), which selects a threshold automatically from a gray level histogram, producing the binary image in Figure 2.11b. Afterwards, a median filter with a size of 10×10 is

applied to help reduce the "salt and pepper" noise and preserve edges. The image is now ready for feature extraction.



(a) Input image (b) *L* channel

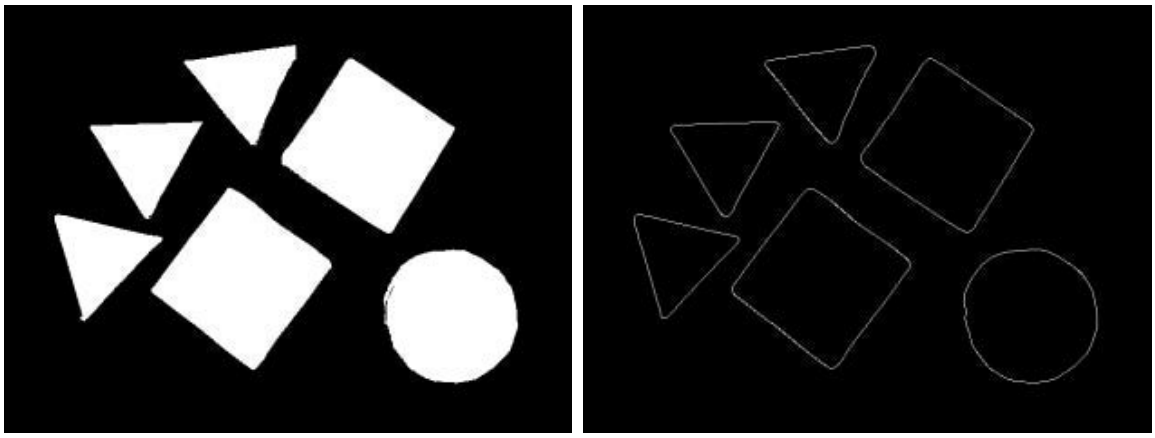
Figure 2.10: Conversion of the input image from RGB to HSL color space (Zakaria, Choon, and Suandi, 2012)



(a) *L* channel (b) Binary image

Figure 2.11: Application of Otsu (1979)'s threshold technique (Zakaria, Choon, and Suandi, 2012)

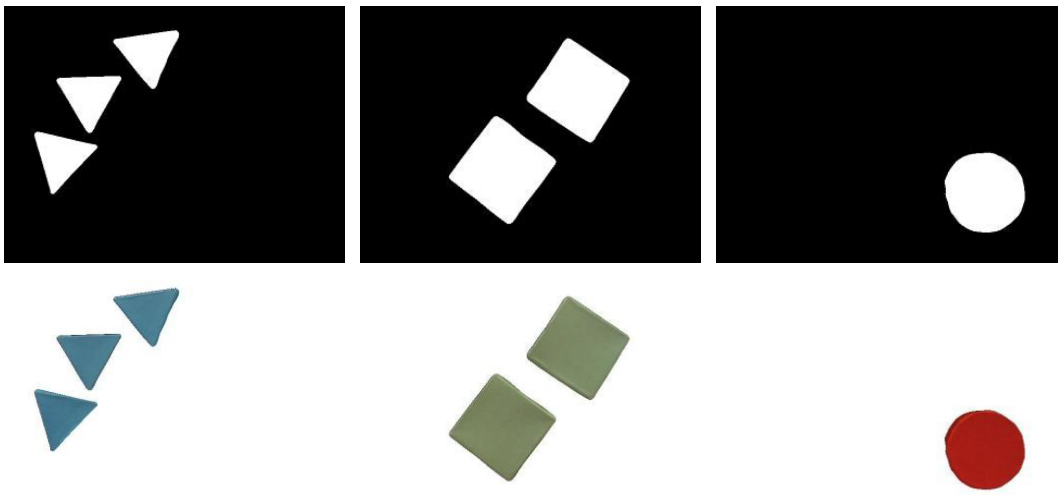
Feature Extraction Having preprocessed the input image into a more simplified, albeit meaningful, version a feature extractor is used to further decompose the image. In this particular case, the Sobel operator (Section 2.1.1.1) is applied upon the binary image in order to identify the set of edges, producing the resultant image shown in Figure 2.12b. Optionally, a thinning operator can be applied to further enhance the edges, especially in cases where there may be a pixel count increase in pixel arrangements that do not form a straight line after the applying the Sobel operator.



(a) Binary image (b) Edges image

Figure 2.12: Application of the Sobel operator (Zakaria, Choon, and Suandi, 2012)

Feature Detection In this stage, the method used to match the features is the Compactness method previously described in Section 2.1.2.2. In this system, Zakaria, Choon, and Suandi (2012) attempt to match the features extracted with three simple shapes – triangles, squares and circles – one at a time, and use these templates to segment the original image through image subtraction. This yields the results shown on Figure 2.13.



(a) Triangle output (b) Square output (c) Circle output

Figure 2.13: Shape detection and segmentation

Since this system is rather simple, it can form the basis for more complex systems comprising of, for example, multiple alternative preprocessing techniques, feature extractors and matching methods.

2.2 Virtual and Augmented Reality in Geometry Education

The ultimate goal of geometry education is to improve one's spatial skills. The idea of using virtual reality (VR) technology to aid in the geometry learning process is not new, and studies conclude that spatial abilities can also be improved through the use of virtual reality (Osberg, 1997). There are two main trending approaches in educational geometry software: the traditional, desktop-oriented application and the increasingly popular, immersion-oriented augmented reality (AR) environments. The latter, while offering a higher tier of immersion and interactivity at the cost of specialized hardware, falls in line with the goals of this dissertation: geometry learning and knowledge consolidation through the use of real world examples in an engaging, exploratory manner. Below is presented an overview of a few relevant systems developed at the time of this writing.

2.2.1 Simple Android Shape Detector

This is the most similar application to the expected contribution of this dissertation. Developed by Kareva (2011a) as an alternative to the costly interactive walls [2], its aim was to make any white wall interactive by sticking papers with different shapes and colors on it, as shown in Figure 2.14, and then detecting them using any smartphone with a camera, as depicted in Figure 2.15a. It was later extended as a memory game showing the player an array of shapes (triangles, quadrilaterals and circles) that he/she would have to remember and then detect on the sticker wall in the same order. It is worth noting that the detection was carried out through a method similar to the Compactness method (Section 2.1.2.2), albeit somewhat more empiric and directed to a specific sticker size.

This application was later tested on a local primary school (Figure 2.15b), with a total of 75 children in 3 different classes, with 34 having access to a smartphone (Kareva, 2011b). Since no assessment was made on the educational quality of this application, as it was not designed nor developed for that purpose, it stands that more research must be conducted concerning the use of mobile devices for geometry learning. Fortunately, the attention grabbing potential seems to be quite evident since the children promptly adhered to the use of this technology in the classroom, which it could prove a good indicator of a successful integration of mobile devices in the learning process.

[2] http://www.exergamefitness.com/twall.htm

Figure 2.14: A white wall sticked with papers of different shapes and colors (Kareva, 2011a)

[3]Taken from http://www.youtube.com/watch?v=zwDVMs0P5jw and http://www.youtube.com/watch?v=YBorTg__olM

(a) Detecting a circle (b) Child using the application

Figure 2.15: The Simple Android Shape Detector[3]

2.2.2 Construct3D

Construct3D (Kaufmann and Schmalstieg, 2002; Kaufmann, 2004; Kaufmann and Dünser, 2007) is a three dimensional geometric construction tool specifically designed for mathematics and geometry education which aimed for the improvement of spatial abilities. The system makes use of AR to provide a simple, easy to learn environment for face-to-face collaboration between teachers and students. The underlying idea is that by using AR technology in detriment of more traditional methods (pen and paper), allowing students to visualize 3D objects and to work directly in 3D space (Figure 2.16), a better and faster understanding of complex spatial problems and spatial relationships can be achieved.

Figure 2.16: Students using Construct3D in a lab setup (Kaufmann and Schmalstieg, 2002)

To accommodate the myriad of possible interaction scenarios in an educational environment, and acknowledging many of them cannot realistically be accomplished in schools (even nowadays) with the same expensive lab equipment comprised of tracking systems, Head Mounted Displays (HMD) and stereoscopic video projectors, Kaufmann and Schmalstieg (2002) devised three "hybrid" hardware setups:

Augmented Classroom Setup This setup consists of two wearable AR kits composed of a back pack computer, stereoscopic see-through HMD with camera and custom pinch gloves for two-handed input. One kit is intended for the teacher and the other for one of the students. It is obvious this setup restricts use in larger groups due to the limited number of available kits. However Kaufmann and Schmalstieg (2002) compare this situation to the use

of a blackboard in class – either the teacher or a single designated student work on the blackboard, while the rest of the students watches or works along on paper. To further mimic this situation, an additional computer and video projector can be used to project a live, albeit monoscopic, video feed of the constructions being worked on by the kit wearers on a projection screen for the rest of the students to watch, as depicted in Figure 2.17.

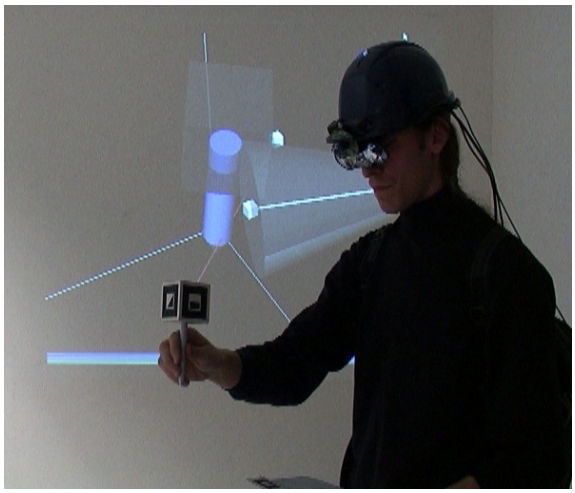


Figure 2.17: A user working in Construct3D wearing the AR kit while a live (monoscopic) video feed of his current construction is displayed (Kaufmann and Schmalstieg, 2002)

Distributed Hybrid Setup In this setup, students are all equipped with a personal virtual reality (VR) system workstation built using a FireWire camera for optical tracking, an of-the-shelf consumer graphics card and a pair of stereoscopic shutter glasses. Students work with hand held props equipped with markers, which are tracked by the camera, and can choose individual viewpoints or be locked to the teacher's viewpoint in a guided mode. It is a relatively low price, personalized setup (Figure 2.18).

Projection Screen Setup This setup is a less immersive but also less complex and expensive option. It is simply a large screen projection shared by the class in which is displayed images of the current user's construction. Although the images displayed are typically stereoscopic, therefore requiring active or passive stereoscopic glasses, there are a number of issues pointed out such as severely distorted images and objects not appearing to be aligned or superimposed with the user's hands.

During the first evaluation of Construct3D, a few points immediately stood out. Despite the many setups devised, the system was at its most fulfilling state

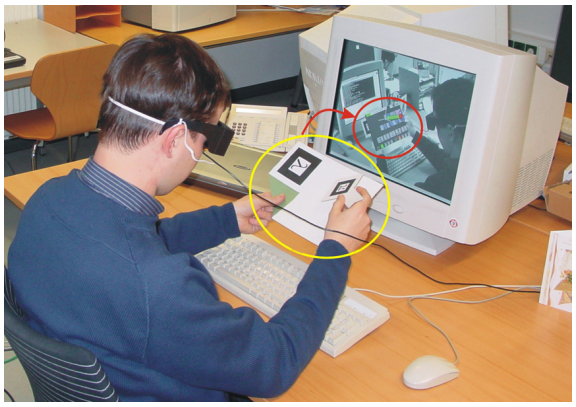


Figure 2.18: The hybrid setup. The yellow ellipse is the hand held prop tracked by the FireWire camera (out of view) and the red ellipse is the VR overlay over the video image

when the lab setup was used, since it had been primarily designed and developed for that hardware. Another point was how easily the students (high school students in this case) interacted with the system, and applied their experience with 2D interfaces to the system's 3D interface (Kaufmann and Schmalstieg, 2002). Most completed the assigned tasks with time to spare and boasted a sense of pride in their constructions, to the point of walking around or admiring them from below. When asked if it felt easier to work with Construct3D for the first time comparing to their first experience with traditional desktop Computer Assisted Design (CAD) software, most students agreed and mentioned that they could imagine themselves working with Construct3D without any previous CAD experience. It is interesting to note that most students outlined the "playful" way of constructing and exploring which helped them focus on the task at hand, and that the knowledge acquired from the experience is easier to recall.

While the key objective for Kaufmann and Schmalstieg (2002) is a tier above this dissertation's, considering that the focus in on 2D geometry learning and targeted at a younger age range, the means are somewhat similar – the use of real-world examples and do-it-yourself exploration to stimulate knowledge consolidation.

2.2.3 TinkerLamp

In an effort to seamlessly integrate conventional school tools in computer interfaces designed to learn geometry and promote group and classroom-level learning, Bonnard and Verma (2012) presented an AR-based tabletop system possessing interface elements made of paper. It was designed for young students under the hypothesis that geometry education in primary schools can benefit substantially from the use of paper interfaces and inherent characteristics. The reasons for choosing paper lie in its persistent, malleable, adaptive and low cost characteristics, strong integration with the classroom environment, and even the fact that many computer interface metaphors such as cut-copy-paste, files and folders, are inspired by practices involving paper.

The system, shown in Figure 2.19, is known as TinkerLamp. It incorporates a camera and a projector directed at the tabletop surface by a mirror, effectively extending the augmented surface area. It sports an embedded computer with minimal interaction (ON and OFF) and relies on the use of fiducial markers to detect, track and bestow meaning to the interface's paper elements, consisting of paper *sheets* and *cards*, while also allowing the possibility of using these very same elements themselves as additional projection surfaces due to the top-down projection mode. Conventional geometry tools such as ruler and protractor are also a part of the system, completing what is referred to as a *scattered interface* (Bonnard and Verma, 2012).

In order to both study the system's educational impact on primary schools and integrate it accordingly in the conventional classroom curriculum, three activities where created, each taking into consideration one of the three circles of usability in the classroom – individual, group and classroom. Video material of the following activities can be found in Bonnard (2012).

Classifying Quadrilaterals The first activity had the students classify a set of quadrilateral cardboard *shapes* into squares, rhombuses, trapezoids, etc. This was accomplished through the use of a set of three marked cards, each with its own function that would activate when placed close to a shape. For example one of the cards would display the shape's angle measures while another would highlight the parallel sides. A combined use of all cards would yield all the shape's basic characteristics, as can be seen in Figure 2.20. A fourth card was used for validation purposes. This specific activity was deployed in schools with students in the 7 – 10 age group and showed a fast adoption by the students alongside a good deal of creativity when using

Figure 2.19: The TinkerLamp system (Bonnard and Verma, 2012)

the system. Actually, the combination portrayed in Figure 2.20 was thought up by a group of students who created this "test bench" and just swapped shapes to observe their characteristics simultaneously (Bonnard and Verma, 2012).

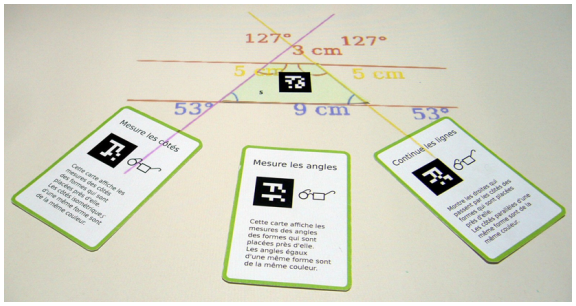


Figure 2.20: The "test bench" displaying all the shape's basic characteristics (Bonnard and Verma, 2012)

Discoverying the Protractor The second activity was designed to help the students learn to use the protractor in an exploratory mindset, after being introduced to angles in the classroom. A deck of marked cards is available, containing two kinds of cards: two *angle control cards* and ten *angle measure cards*. These are further divided into two groups based on the orange or blue icon printed on them. Orange colored cards indicate that a given angle is to be measured in a *clockwise* direction while blue colored cards denote an anti-clockwise measurement. During the collaborative development of the activity with school teachers, this distinction was identified as the main difficulty when learning to use the protractor. Each angle measure card has a different angle value printed on it and falls upon the student to use the same colored angle control card to reproduce the angle in the respective direction. Once the student is confident that the angle is correct, he/she flips the angle measure card to validate the result. Figure 2.21 presents an example.

This activity was conducted with over 100 students in the 8 – 10 age group, which were divided into groups and carried out the measurements. Afterwards, the students undertook a test on paper where they were asked to identify and write down a series of angle measures next to a printed protractor. Most students had a strong engagement response while using the system to complete the given task and the creativity surfaced once again, with students switching off the feedback for the sake of suspense. However the tests did not yield any significant results due to a ceiling effect (Bonnard and Verma, 2012).

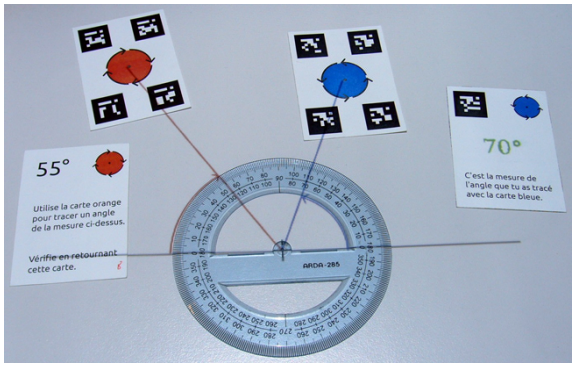
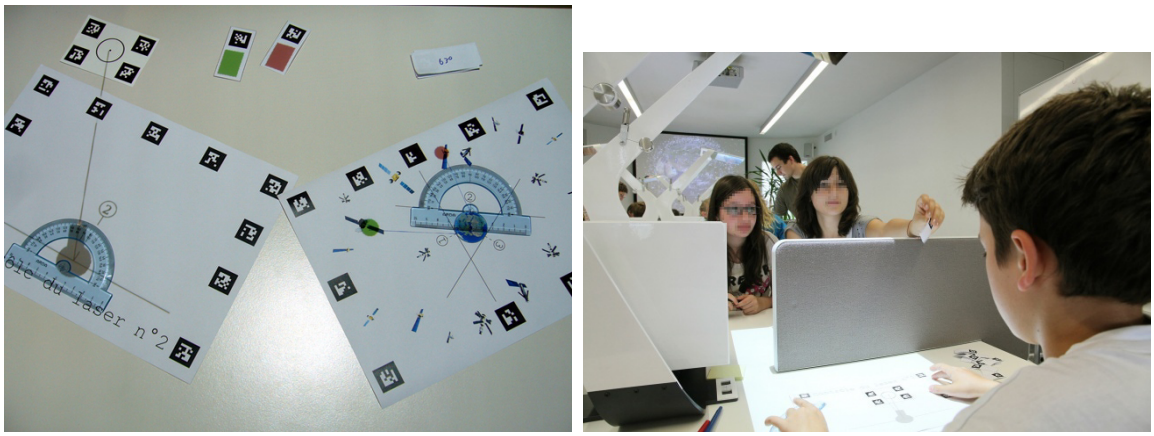


Figure 2.21: Measuring angles in both clock and anti-clockwise directions (Bonnard and Verma, 2012)

Describing Angles Yet another activity with game-like traits, its objective was to have students use a laser gun to destroy space junk, non-functional satellites still orbiting around Earth (Figure 2.22a). The educational goal was to develop the students' angle describing skills by being able to convey clearly to someone an angle measure, direction of measurement (clock or anti-clockwise) as well as the most convenient axis reference. A group of four students were divided into two cooperating teams, *observers* and *controllers*, and a physical separation was set between them (see Figure 2.22b). The first were in charge of choosing the next target and were granted a sheet with all the satellites as well as the position of three laser guns along with their respective axis printed on it. Using a protractor, they would pick a suitable laser gun axis and measure the angle to the chosen target satellite and describe it to the controllers who were standing by. The latter were provided three sheets corresponding to the 3 laser guns (seen in the left part of Figure 2.22a), and would use an angle control card (similar to the one used in the previous activity) to change the inclination of the desired laser gun in order to reproduce the angle given by the observer team. Once confident that the angle reproduction was accurate, they would open fire by flipping another card representing the laser's ammunition. Being the ammunition finite, the students would have to exercise caution instead of simply employing trial-and-error strategies.

In explaining the activity to the over 140 students who participated, there was an intentional omission about what information was required to describe the angle, leaving the students to figure that for themselves. The main difficulty of this activity was to establish a convention to describe and communicate an angle without seeing it (Bonnard and Verma, 2012). Besides the obvious measurement requirement, the students quickly realized the origin of the shot (which laser to use) was also needed. Having laid down a convention, the progress was smooth, with some students preferring to move the sheets around in order to use the same referential every time. However, some students struggled with the direction of the shot, but this served as an opportunity for the teacher to intervene and review the clock and anti-clockwise concept.



(a) The controllers (left) and observers (right) sheets

(b) The teams physically separated by a barrier

Figure 2.22: The space junk satellite cleaning game (Bonnard and Verma, 2012)

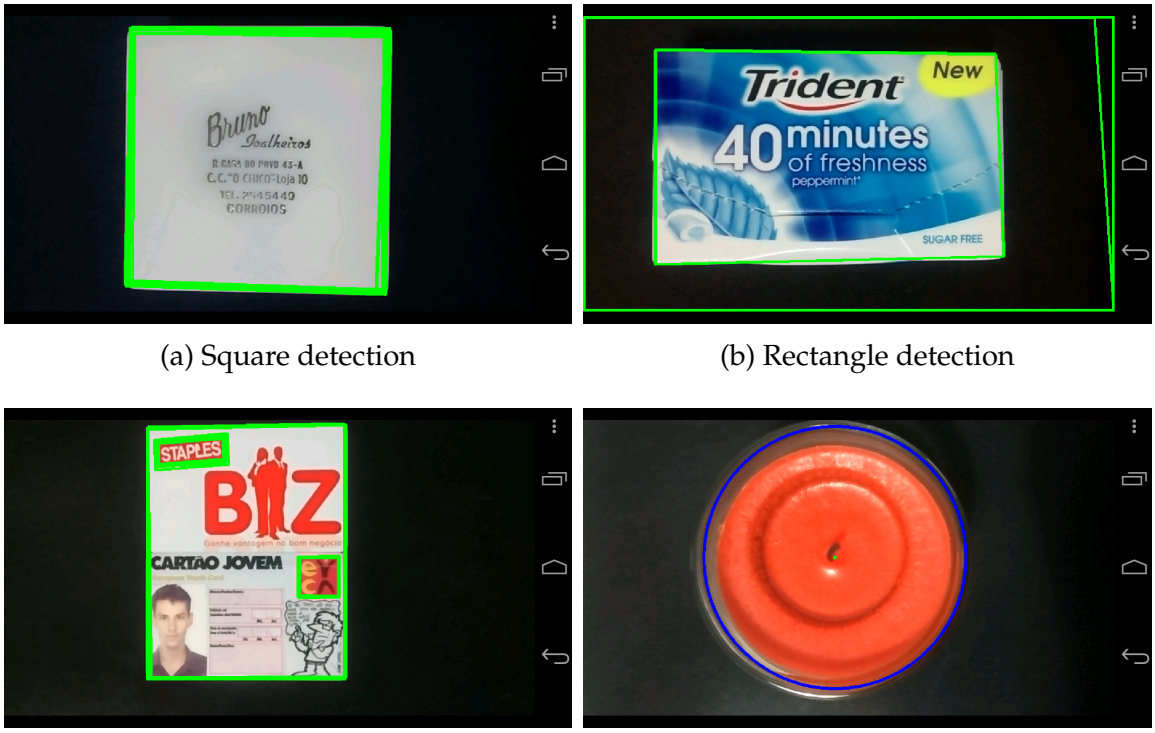
2.2.4 Discussion

All the projects above demonstrated to some extent that children are quite receptive to the idea of using VR methods to complement their geometry learning process, and do so in a very engaging manner while exercising their creativity. This in turn promotes their productivity, sense of self-accomplishment and, more importantly, better consolidates their knowledge.

Although the last two projects (Kaufmann and Schmalstieg, 2002; Bonnard and Verma, 2012) involved the use of the somewhat access-limited AR technology, their results and achievements provide a source of inspiration and the base for new ideas. However this dissertation will follow along the path laid down by Kareva (2011a) and use the appeal of the common yet exciting smartphone (and its capabilities) to attract the children's attention, curiosity and creativity for educational purposes.

2.3 Exploratory Application

An exploratory application was developed with the purpose to assess the performance of readily available shape detection algorithms found within OpenCV sample code. It is able to detect squares, rectangles and circles. The developed testbed was a Samsung Galaxy S with a 1GHz single core processor, 512MB of memory and a 5MP camera running Android 4.3 Jelly Bean. Figure 2.23 below shows the application in action.



(a) Square detection

(b) Rectangle detection

(c) Quadrilaterals detection

(d) Circle detection

Figure 2.23: The small exploratory application

As can be seen in Figure 2.23b, when detecting rectangles the application ended up detecting the camera edges as well. While technically correct, this was not intended. Another thing to note is the use of two adjacent rectangles to form a square in Figure 2.23c. This hints at the possibilities children have to conjure up solutions to answer the game's questions (*i.e.*, to show a square made of other shapes).

What cannot be conveyed through figures however, is the toll this detection took on the smartphone resources. While detecting, the camera frame rate was reduced below 5 FPS (Frames Per Second), which made the experience feel "choppy" and downright unacceptable as a game. This was due to the use "as is" of provided OpenCV detection sample code, which is more desktop-oriented. Besides

the obvious raw processing power compared to mobile devices, desktops have another, rather unexpected upside: most real-time detection is performed using webcams which usually have relatively low resolutions, whereas mobile devices' cameras evolve at a startling pace and is not uncommon to come across FullHD resolutions at the time of this writing. Another tested hypothesis was whether the Android OS (Operating System) itself can have an impact on performance. From the time when these tests where conducted, Google released Android 4.4 Kitkat which it claimed to be optimized for devices with as little as 512MB of memory – 4.3 required a minimum of 1GB (Google, 2013). However this did not have any positive impact as far as the application is concerned and further presses the need for more lightweight algorithms.

3

Mobile-Oriented Library

In this chapter the transition is made from the techniques and systems studied in Chapter 2, along with the small exploratory application results, into the mobile-oriented library requirements and functions. Sections 3.1 and 3.2 allude to the requirements this library strives to meet and its structure. Section 3.3 details how the Detector module actually performs detection. Finally, Section 3.4 briefly explains the purpose of the Painter module.

3.1 Library Requirements

Given the inefficient results obtained from the OpenCV sample algorithms, past experience with mobile devices and some common sense, three major requirements were outlined prior to the library development: be lightweight, adjustable and portable.

Lightweight Due to the substantial frame rate drop the first major requirement was coincidentally to be able to perform shape detection while sustaining a reasonably high frame rate at native resolutions, or close-to-native as some devices' cameras output unreasonably high resolutions such as 4K which current mobile processors simply cannot keep up with regarding real-time shape detection. While this can have an impact on the accuracy of the detection itself, translated in having the least amount of extraneous operations on any given frame, it is expected that, given the relative simplicity of the

shapes to be detected, satisfying results can be yielded.

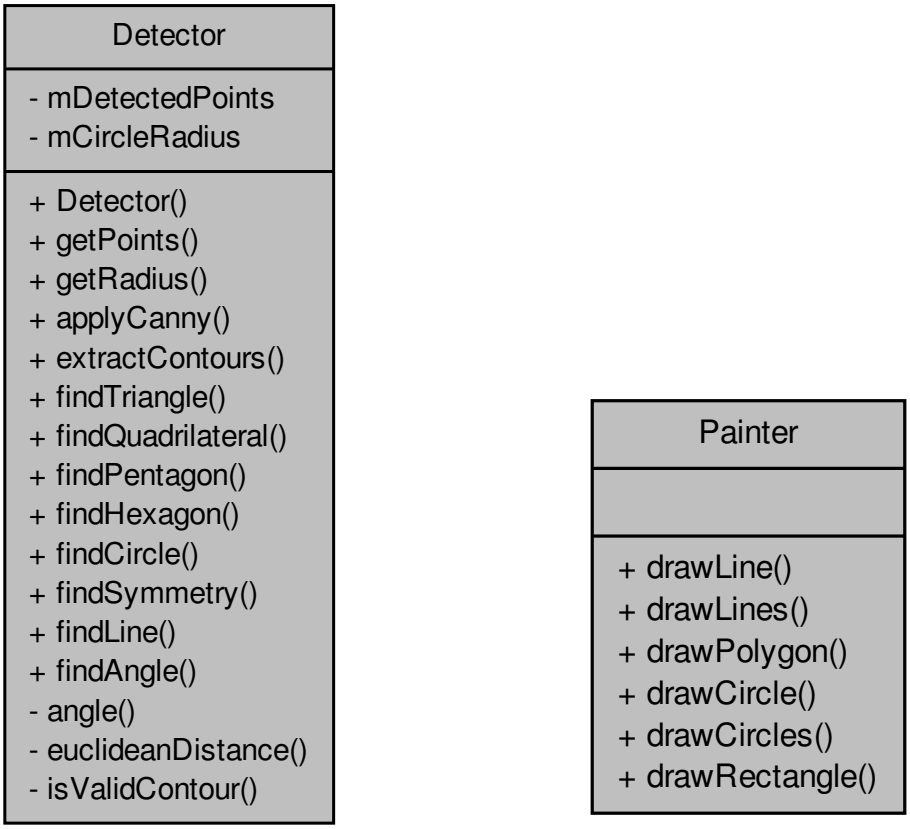
Adjustable Being able to adjust options and parameters is the hallmark of any proper library. Therefore this library is to provide a few customization options without straying too far for the intended purpose.

Portable Despite being a mobile-oriented library, this does not mean it has to be mobile-exclusive. The library should be able to be used within multiple environments where OpenCV is available and to perform shape detection on any frame, regardless whether it comes from a video stream or still imagery. While OpenCV provides a Java interface to it's functions that integrates nicely with Android, this proves very limiting, outright excluding not only desktops but also other mobile platforms such as iOS. Development in C++ both solves this impedance and offers performance benefits on the Android platform itself. More on this in Chapter 4.

3.2 Library Structure

This library is comprised of two independent, yet interoperable, components: the Detector and the Painter. The former is the core component – it is there that all the heavy lifting is done – and the latter is used to display the detection results on the frame, providing different ways to do so. As mentioned previously, it is written in C++, leveraging the functions available in OpenCV. An overview of the library structure can be visualized in Figure 3.1 and Tables 3.1 and 3.2.

The Detector class, `detector.cpp`, provides several functions that operate on a given frame to produce a set of points representing the detected shape. These points can then be passed on to the Painter class, `painter.cpp` in order to draw said shape onto the frame. This decoupling of the Detector and the Painter allows for extra flexibility, as the points can be used for whichever purpose the user deems appropriate – painting is simply an option.



(a) Detector class diagram (b) Painter class diagram

Figure 3.1: The library structure

Function	Description
`getPoints()`	Returns the points of the last successful detection
`getRadius()`	Returns the radius of the last successful circle detection
`extractEdges()`	Finds the edges of a greyscale frame
`extractContours()`	Extracts contours of a binary image
`findTriangle()`	Analyses the contours for triangle presence
`findQuadrilateral()`	Analyses the contours for quadrilateral (rectangle or square) presence
`findPentagon()`	Analyses the contours for pentagon presence
`findHexagon()`	Analyses the contours for hexagon presence
`findCircle()`	Analyses the contours for circle presence
`findLine()`	Analyses the contours for parallel/perpendicular line presence (in respect to the y-axis)
`findAngle()`	Analyses the contours for acute/right/obtuse angle presence
`findSymmetry()`	Analyses the contours for symmetry presence (in respect to the y-axis)
`angle()`	Returns the cosine of the angle between three points
`euclideanDistance()`	Returns the Euclidean distance between two points
`isValidContour()`	Checks if a contour is valid within the application's context

Table 3.1: Detector functions overview

Function	Description
`drawLine()`	Draws a line between two points
`drawLines()`	Draws a line between each set of two points
`drawPolygon()`	Draws a polygon (open or closed) from a set of points
`drawCircle()`	Draws a circle with a specified centre and radius
`drawCircles()`	Draws several circles with a specified radius at different centres
`drawRectangle()`	Draws a rectangle from a Rect object

Table 3.2: Painter functions overview

3.3 Detector

It is worth recalling that a video is composed of a sequence of image frames, which in turn are simply pixel matrices. They have a number of columns and rows, pertaining to the width and height, and can have as many layers as the frame's channels. Greyscale 8-bit frames have a single layer, while RGBA 32-bit have four layers. Given this notion, it is also reasonable to think of a part or region of a frame as submatrix of the original matrix – a Region of Interest (ROI) (Laganière, 2011). Therefore it is more accurate to state that the Detector operates on matrices, specifically greyscale matrices, abstracted from whether it is the whole frame or simply part of it. However, for the sake of simplicity, the terms *frame* and *ROI* will be used instead of matrix and submatrix. The reason for choosing greyscale is performance and efficiency related: not only is it faster and simpler to work on a single channel, but most times better feature extraction (and by extension better results) can be achieved by using such frames, as per Chapter 2.

Next will be discussed what can be currently detected and how the actual detection is performed. As can be seen in Figure 3.1a, detection can be split into two categories: geometric shapes and geometric notions. The geometric shapes that can be detected are triangles, quadrilaterals (rectangles and squares), pentagons, hexagons and circles, while the geometric notions that can be detected are lines (parallel and perpendicular), angles (acute, right and obtuse) and axis symmetry (horizontal and vertical). Several reasons are behind this detection set. The technological standpoint is the more complex the shape, the more computationally expensive becomes detecting it. The practical standpoint lies with the fact that there aren't many geometric shapes past hexagon occurring in the real world. One other reason falls on the target audience of this study – children in the 4th grade usually only learn as far as the hexagon. Despite that, the Detector can be easily extended to support more complex geometric shapes.

3.3.1 Extracting Features

As recalled by Figure 2.9 the first step in detection is preprocessing the frame. Since the frame is expected to be already in greyscale, this step is a blur to minimize the noise. The extracted features are the frame's contours. Corners were also considered, but dropped in favour of contours for they can convey useful information such as the shape's convex property. This is accomplished in two phases, through the functions `extractEdges()` and `extractContours()`.

3.3.1.1 extractEdges()

This function takes the greyscale frame and outputs a binary frame representing the edges. The edges are extracted by two alternative approaches: an adaptive threshold or the Canny operator. The adaptive threshold employs a *mean-C* strategy (Fisher et al., 2003) to better deal with illumination gradients. It allows an adjustable *kernel* size which defaults to 11 and a *C* constant which defaults to 2. On the other hand the Canny operator applies hysteresis thresholding with the high threshold being calculated through the method by Otsu (1979) and the low threshold being set to half of that value.

$$\begin{aligned} T_{high} &= Otsu(frame) \\ T_{low} &= T_{high}/2 \end{aligned} \tag{3.1}$$

Whichever approach is used, both are preceded by a either a Median or a Gaussian blur (again with a adjustable *kernel* size, defaulting to 5) in order to enhance edges. While the Median blur is more adequate at enhancing edges, it also is more computationally intensive than the Gaussian blur, and under certain circumstances both can produce virtually the same result, hence the resolve to add both as an option.

3.3.1.2 extractContours()

After obtaining the binary frame containing the edges, it is time to extract a representation of any meaningful contours, which could be potential geometric shapes. The `extractContours()` achieves this through the use of the OpenCV function `findContours()`, which takes in the binary frame and outputs a list of contours. This function can be adjusted to deliver all contours with no particular relationships, in an hierarchical fashion distinguishing between outer an inner contours, or only the outermost contours, which is the default mode since it is most suited for geometric shape detection as there isn't any particular concern regarding holes or concentric shapes.

3.3.2 Matching Features

Once the contours are extracted it is time to analyse them in order to try and match them to a given geometric shape or notion. All geometric shapes follow the same initial pattern and differ slightly in the analysis stage, unlike geometric

notions, with each notion going down its own route.

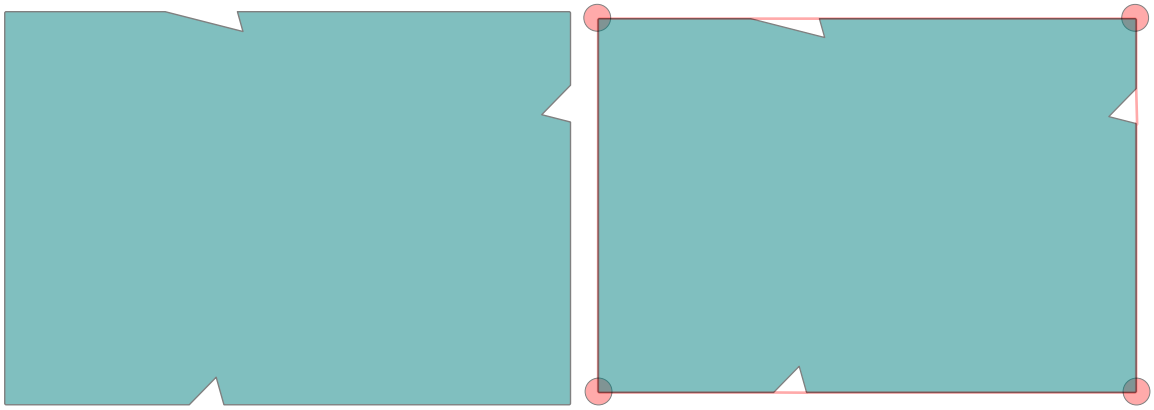
In this common initial pattern all contours are subjected to a validity check, requiring the contour area to make up at least 10% of the frame area, as per Equation 3.2, and be convex – a requirement for basic geometric shapes – or else be immediately discarded.

$$Area_{contour} >= Area_{frame} * 0.1 \tag{3.2}$$

Should a contour meet the area requirement, it is then checked for a convex property. In order to do this, it is subjected to another treatment common to all geometric shapes: a contour approximation. The reason for this is twofold. First it is impracticable and inefficient to work with the hundreds, possibly thousands of points that comprise a contour, therefore the contour must be abstracted into something that still conveys the same meaning while being substantially easier to work with. Second, not all objects in the real world are perfect depictions of geometric shapes. A rectangle could have a small defect on one of it's sides or rounded corners, yet the human eye can still perceive it as a rectangle. The approximation can overlook such defects and caveats, essentially mimicking the human eye to an extent. OpenCV provides a function for that called `approxPolyDP()`, which is based on the algorithm by Douglas and Peucker (1973). It approximates a curve or a polygon with another curve/polygon with less vertices so that the distance between them is less or equal to a specified precision. This precision is what determines the approximation result. Too high and small defects will not be compensated, too low and serious defects will be overlooked. The value being used is based on the contour perimeter, calculated with another OpenCV function called `arcLength()` with a 2% deviation following Equation 3.3. Using this small deviation allows for a rather rigorous approximation of the original contour while compensating small defects.

$$Precision = Perimeter_{contour} * 0.02 \tag{3.3}$$

An example can be seen in Figure 3.2.



(a) Contour (b) Approximation

Figure 3.2: Application of the approxPolyDP() function

If the contour in Figure 3.2a was not approximated, it would be discarded as not being convex, while it clearly resembles a rectangle. Approximating the contour results in Figure 3.2b. It has transformed the defect-ridden contour into a rectangle, or more precisely the four points that comprise the approximated rectangle's vertices.

The first requirement excludes many small, usually noise-related, contours and the second non-convex shapes, which in turn speeds up the detection process by focusing only on potential geometric shapes. Once both requirements are met, the contour – now turned into an approximation – is deemed valid and added to an array of relevant approximations which will be analysed in order to ultimately decide whether a specific geometric shape can be found in the frame. The functions below detail how this analysis stage is performed for each geometric shape.

3.3.2.1 findTriangle()

Triangles are detected either through their internal angles or relative length of sides. For example, a right triangle has one internal angle measuring 90^{o} and a scalene triangle has no two sides with equal length. Although equilateral, isosceles and scalene triangles are more commonly tied to their side properties, each also possesses unique characteristics regarding its internal angles. Therefore all types of triangles can be detected through internal angle analysis. This function iterates the approximations array performing a final check before the actual analysis – the approximation must have exactly three points. If the former holds true, then it takes all three points and cycles the angle point, measuring it and then going down one of the cases below:

Equilateral All internal angles measure close to 60º.

Isosceles Two internal angles have roughly the same measure.

Scalene No two angles are the same measure.

Right One angle measures close to 90º.

Obtuse One angle measures higher than 90º.

If an approximation is found to comply with the case respective to the attempted detection, it is marked as a triangle.

3.3.2.2 findQuadrilateral()

Quadrilaterals can also be defined by their internal angles all measuring 90º. Therefore the procedure is relatively the same: check for four points first and then cycle them in triplets measuring the angles. If all angles are close to 90º all that is left is checked whether the quadrilateral is a rectangle or a square. This is accomplished by applying the Euclidean distance in order to measure the width and height and then compare them. If equal, the approximation is a square, else it is a rectangle.

3.3.2.3 findPentagon()

Regular pentagons have all internal angles measuring 108º. If the approximation has five points and all angles measure close to 108º, then the it is marked a pentagon.

3.3.2.4 findHexagon()

Regular hexagons have all internal angles measuring 120º. Again, if there are six points and all angles measure close to 120º, then the approximation is marked a hexagon.

3.3.2.5 findCircle()

Finding circles in frames is usually accomplished through the use of the Hough transform (Duda and Hart, 1972), with very good accuracy. Unfortunately, the accumulator space and voting system inherent to the algorithm exert a great strain on mobile devices and the frame rate drop is too steep, hence it is not a viable approach for real-time circle detection. On the other hand, having limited the

geometric shape detection up to hexagon opens the possibility for a substantially faster approach in circle detection. In essence, when approximating a circle with the precision specified previously, the approximation will still contain a large number of points. Taking advantage of this, any approximation with a number of points larger than a certain value is marked as a circle. As it stands, this value is defined as 10 points, to avoid accidental marking of octagons as circles.

This concludes the geometric shape detection functions. A few points must be discussed before moving on to geometric notions. First it is mention throughout the functions the expression "close to X^{o}" This is again tied to the fact that most real world objects are not perfect depictions of regular geometric shapes, and can be further distorted when captured through different camera resolutions and subjected to feature extraction. A perceived square can have its internal angles deviating slightly from 90º, say by $\pm 1.5^{o}$. Ergo, it is reasonable to relax some of these properties, and to use acceptance margins. Another point is how the angles are actually measured. The cosine of the angle is the actual measure being calculated, as per Equation 3.4.

$$\frac{(dx_1 * dx_2 + dy_1 * dy_2)}{\sqrt{(dx_1^2 + dy_1^2) + (dx_2^2 + dy_2^2)}} \tag{3.4}$$

Where dx_1 and dx_2 is the x distance between the each point to the angle point and dy_1 and dy_2 is the y distance. The next point is performance driven. Starting with quadrilaterals, there is a property that allows for a small speed up: if three angles measure 90º, then the last angle must also measure 90º. Generalizing, for regular geometric shapes with more than three sides, if $n - 1$ angles are equal, then angle n must also measure the same. Thus, there is only need to measure $n - 1$ angles to verify the existence of a regular geometric with n sides.

The Compactness method, described in Section 2.1.2.2, was employed at some point to detect squares and circles, bypassing any angle measuring. In summary, assuming a square was the shape sought after, any four-sided approximation would have its perimeter and area calculated and subsequently used in Equation 2.3 to obtain the compactness value. With circles, a bounding rectangle would be placed on the approximated contour and have its compactness value calculated. Bounding rectangles on circles are actually bounding squares (Figure 3.3), and the method relied on this property. If the compactness value was within a certain margin of the respective value (16 for both in this case) then the shape would be accepted. While this method worked with a satisfying success rate, the angle and size measuring approach proved more resilient to perspective

skewing and distortion.

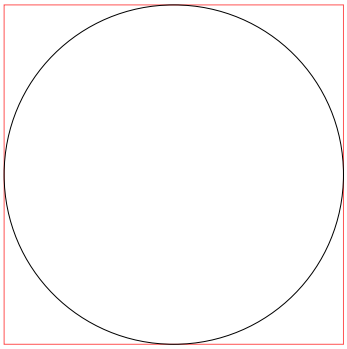


Figure 3.3: Bounding property for circles

Lastly it is important to remember that whenever a detection is successful, the approximation points are effectively what is returned to the caller, through modification of the array variable `mDetectedPoins`. With circles, the array contains only the centre point and the variable `mCircleRadius` is updated with the circle radius, which are acquired by placing a bounding rectangle (more accurately a square in this case) over the circle and calculating the centre of the square and half its height (Equations 3.5 and 3.6), respectively.

$$\begin{aligned} x_{centre} &= x_{topleft} + height/2 \\ y_{centre} &= y_{topleft} + height/2 \end{aligned} \tag{3.5}$$

$$radius = height/2 \tag{3.6}$$

3.3.2.6 findLine()

This function detects lines in a frame which obey a parallel or perpendicular specification. It takes the edges frame as input, an angle and a boolean value dictating whether it should look for a parallel (true) or perpendicular (false) line. The OpenCV `HoughLines()` function is used to search for lines in steps of $\pi/2$, in respect to the unit circle, essentially limiting the search to horizontal and vertical lines. Searching only for two directions is much less exhaustive and thus faster enough to make the Hough transform viable for real-time detection. All horizontal and vertical lines are stored in a array and then iterated over, extracting their angle and marking or discarding the line according to the following conditions:

- Input angle equals line angle and boolean is true – parallel

- Input angle equals line angle and boolean is false – discard
- Input angle differs from line angle and boolean is true – discard
- Input angle differs from line angle and boolean is false – perpendicular

Currently only horizontal and vertical lines are searched for, so this equal-or-different logic is enough to categorize them.

3.3.2.7 findAngle()

The initial pattern of this function is somewhat similar to the geometric shapes' pattern, albeit with some key differences. First, the validity check requires the contour length to be at least two times the size of the frame width or height, whichever is smaller. Second, the contour approximation does not require the contour to be closed, nor it is checked for a convex property. An approximation failing these requirements is discarded. Another meeting them yet having only two points is also discarded. The rest of the approximations are then analysed in a similar fashion as the geometric shapes, with triplets of points being cycled for angle measurement. However, should the distance between the one outer point and the angle point be smaller than a certain value the triplet is skipped and discarded. This avoids cases where an approximation has enough length to meet the validity check, yet the triplet being currently analysed is exceedingly close together, making it hard to perceive the angle. After measuring the angle using the Equation 3.4 and converting to degrees, the angles are marked as follows, depending, again, on the type of angle being detected in the first place:

Acute Angle is between 15º and 79º.

Right Angle is between 88º and 92º.

Obtuse Angle is between 101º and 165º.

The reason why these values where chosen is that when detecting angles below 15º it became less pronounced and visually awkward, although technically correct. The margin to 90º from either side is to avoid the grey area where an angle measuring 88º degrees can be perceived as acute by one individual (and it should be, in a perfect depiction) yet perceived as right by another. These margins take the results aways from this grey area into a more explicit zone.

3.3.2.8 findSymmetry

This function currently detects symmetrical, non-curvilinear shapes along a vertical or y axis, placed at the middle of the frame. It follows the similar initial pattern regarding contour approximation, but forgoing the convex property. Symmetry detection is performed by iterating over the approximation vertices and following the steps below:

1. Check if the point is close to the symmetry axis. If yes move to next vertex. If not abort.
2. Search for a vertex with the same y coordinate and at same absolute distance from the symmetry axis. If found move to next vertex. If not found abort.
3. If not aborted, shape is at least y-symmetrical.

It is important to note that, when analysing the array of contour approximations, vertices in a single approximation array are ordered clockwise starting with the vertex closer to $(0, 0)$. This means that in a y-symmetric shape, for every vertex on the left side of the symmetry axis, its corresponding vertex will be on the last half of the array, even if the shape has an odd number of vertices. Putting it another way, for an array of n vertices belonging to a y-symmetric shape, all vertices from $n/2$ (rounded up) to n are either on the symmetry axis or to the right. Knowing this, only the first $n/2$ vertices need to be iterated in either direction to check for symmetry. The corresponding vertex will be searched backwards starting from the end of the array. This speeds up the process for more complex shapes with a significant number of vertices.

Much like the geometric shapes' case, the variable `mDetectedPoins` is modified with the detection result, if successful. The approximation points are returned when performing symmetry detection, whereas for the angles and lines,three points that make up the former and two points for the latter.

3.4 Painter

The Painter's function wrap around OpenCV drawing functions to accommodate input in the form outputted by the Detector – an array of points, plus radius in case of circles – and are pretty self-explanatory. Therefore the Painter is provided as an independent complement to the Detector simply for the sake of convenience, which means the user can make use of it as a ready-made tool, improve it to create her own drawing solution or simply avoid it altogether if she

doesn't require drawing at all. For instance, instead of drawing the edges of a given geometric shape or notion, it can me more useful to visualize only the vertices as circles (symmetry), or perhaps both simultaneously (angles). This module allows to use the same set of points obtained from the detector to draw any of the combinations above without requiring much effort from the user.

4

System Prototype

This chapter details how the mobile-oriented library depicted in Chapter 3 is integrated into the system prototype developed, as well as the system's architecture and components. Section 4.1 walks through the system's architecture. Sections 4.2 and 4.3 show the user interface and some examples of the prototype in action across detection modes. Lastly, Section 4.4 comments on some particularities regarding the state of development.

4.1 System Architecture

The prototype developed has a component-based architecture, effectively abstracting and separating different concerns to different components, as depicted in Figure 4.1. There are two components: the interface/capture Android Java component and the Canvas C++ component integrating the mobile-oriented library. Both components intercommunicate through the Java Native Interface (JNI) framework. Figure 4.1 also illustrates a typical cycle of the operating prototype: the device camera feeds a BGRA frame to the Android component which routes it through the JNI, alongside a greyscale conversion, to the Canvas/Detector/Painter in order to analyse the specified ROI and draw any detected shapes on the original frame. Afterwards the possibly altered BGRA frame is returned through the JNI to the Android component which finally displays it on the device screen.

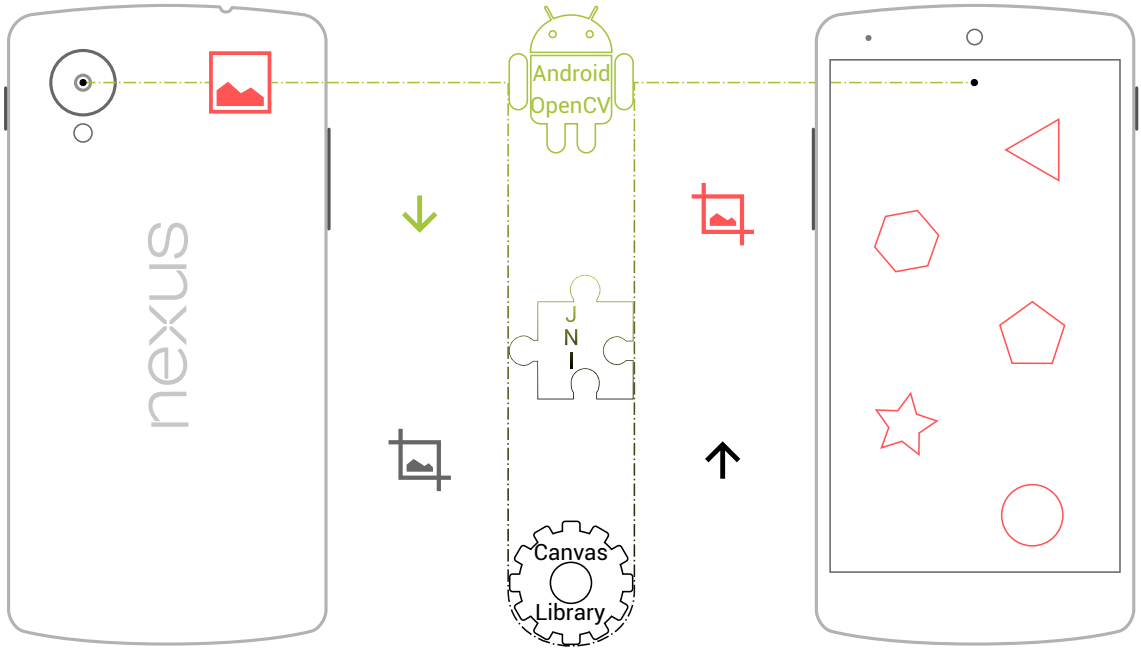


Figure 4.1: System architecture. The mobile-oriented library integrates with the Android Java application through the JNI framework

4.1.1 Android

As stated above, this component is responsible for the interface related work, such as displaying the UI, responding to touch events and changing detection modes, as well as communicating with the device's camera to capture the video frames to be exchanged with the Canvas component or change capture resolution, and running the application as a whole.

In order to access the individual video frames from a given capture, this component makes use of an OpenCV Java library, OpenCV4Android [1], which abstracts having to deal with camera buffers and allows executing code on a per-frame basis.

4.1.2 Canvas

The Canvas component is written in C++, albeit not part of the mobile-oriented library in Chapter 3. Its main goal is coordinating the Detector and the Painter, feeding frames obtained from the camera to the former for detection and using the latter for drawing purposes if so required (*i.e.* detection was successful). Moreover, it also coordinates other aspects in tandem with (and response to) the Android application's life cycle. These aspects involve:

[1] http://opencv.org/platforms/android.html

- Creating and destroying Detector and Painter instances according the application's life cycle stage
- Handling camera resolution changes
- Setting and displaying the ROI where detection will be focused, and resetting upon resolution change
- Setting the detection mode
- Controlling the detection's sliding window
- Setting how results are drawn, full contour vs vertices only

4.1.3 JNI

Putting it simply, this component acts as the glue between the Android Java code and the library's C++ code within the application prototype. While OpenCV is developed and compiled in C++. There are two ways to reach OpenCV on an Android application: through OpenCV4Android or the Android Native Development Kit (NDK).

OpenCV4Android provides a Java API that wraps around the C++ functions and allows calling them directly from Android code space, which abstracts the user from dealing with the native C++ implementation, where computation takes place, and allows for less code separation. However this approach has a downside. Each Java API call will perform a call through JNI to the respective OpenCV function, incurring a non-negligible overhead. This JNI overhead occurs twice: once at the start of the call and again during the return. Now this is not a problem for a single Java API call on a frame (*i.e.* a greyscale conversion). Should another Java API call take place (for instance a blur on the newly converted greyscale frame), then number of JNI overheads is increased to four. A third Java API call would further elevate this number. Thus, the more Java API calls are made, the bigger the cumulative performance penalty. Figure 4.2 exemplifies this.

A slightly more difficult but more performance optimized development method uses the Android NDK. In this approach, the OpenCV vision pipeline code is written entirely in C++, with direct calls to OpenCV. The difference lies in being able to call multiple OpenCV functions by going through JNI just once. Using the same examples as above, the greyscale conversion, blur and the third operation can all be performed on the frame within a single JNI trip, as Figure 4.3 shows, resulting in a noticeable performance gain.

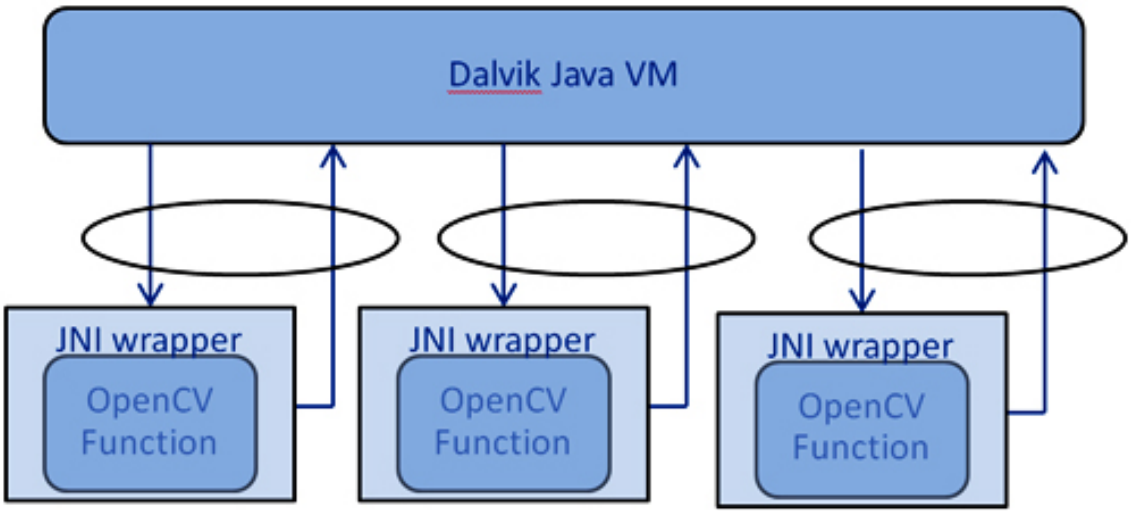


Figure 4.2: Multiple Java API calls incurring multiple JNI overheads

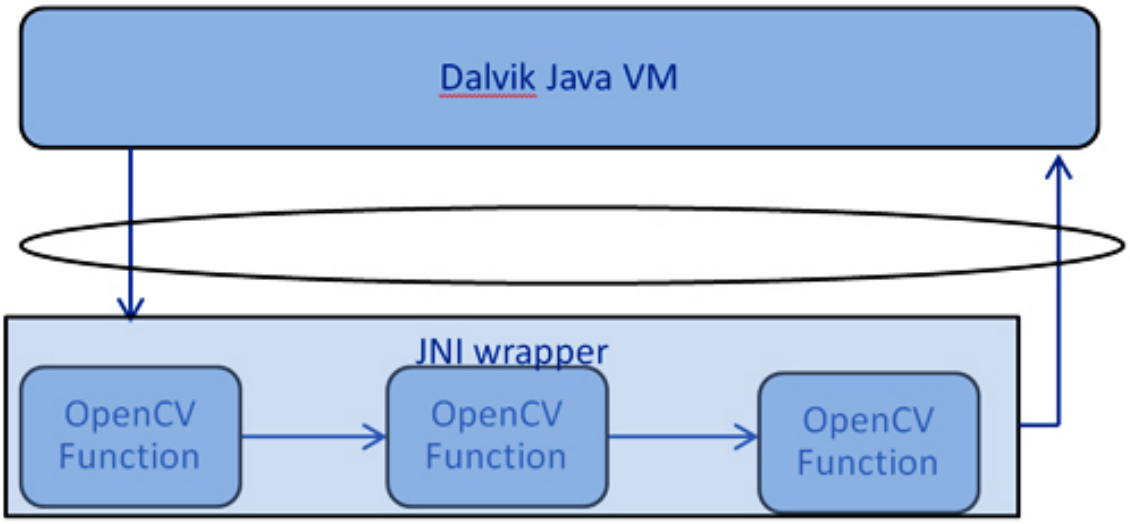


Figure 4.3: Multiple OpenCV calls incurring a single JNI overhead

This approach also allows for code portability, a requirement established in Chapter 3, since the OpenCV algorithms can be developed and tested on a host platform and then moved into different mobile projects.

4.2 Interface

The interface chosen for this prototype, seen in Figure 4.4 is very straightforward. It is comprised of four buttons, one to change between detection modes; one to activate timer mode for performance measuring purposes; one to save an image of the current frame; and one to change video resolution being outputted by the camera. The purpose behind this interface is to give the user quick and easy

access to all prototype functions – everything is at most two taps away – and to get out of the way when not required. After three seconds of non-interaction the interface will hide, allowing the user to focus solely on the viewfinder. To show it back, a single tap anywhere on the screen is all that is necessary. It also possess some degree of opacity, so the user can still visualize the viewfinder when interacting with it.

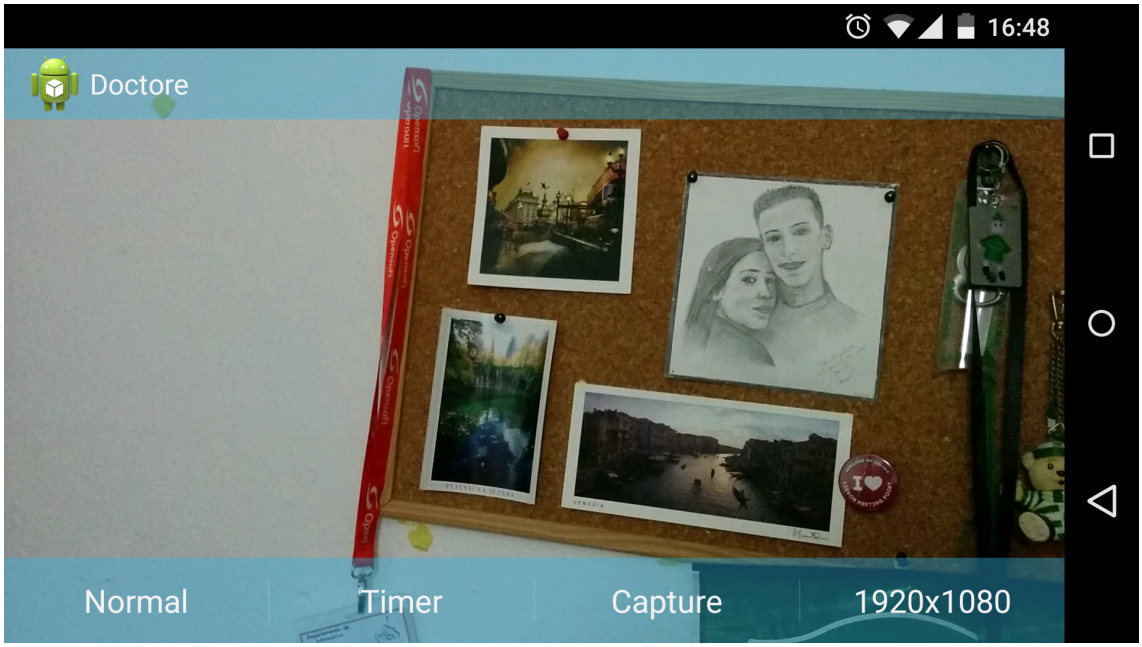


Figure 4.4: The prototype interface. The name Doctore refers to the trainers of gladiators in ancient Rome and much like in that era, it symbolises the hard learning process and journey towards greatness – obtaining a Masters degree

4.3 Detection Modes

Detection modes mirror the library's current capabilities, and add two more modes: (1) a normal camera-to-display feed and (2) an edges mode to better visualize what the device is "seeing", in a sense. When in any of the proper detection modes, detection is only carried out after the device is deemed stationary for n seconds (default is 1) by checking its accelerometer and gyroscope if available. This helps detection accuracy by minimizing motion induced defects and corruption, lowers battery drain while the user searches for an object to present to the camera and avoids accidental, unintended detection in the process. Figure 4.5 shows the mode selection screen and Figure 4.6 a triangle detection with the timer mode on. The four white dots in the frame represent the ROI where the user must

place the object being detected. As stated above, once the device is deemed stationary the dots will turn yellow signifying detection is being attempted. If timer mode is active, the duration of the detection will be logged.

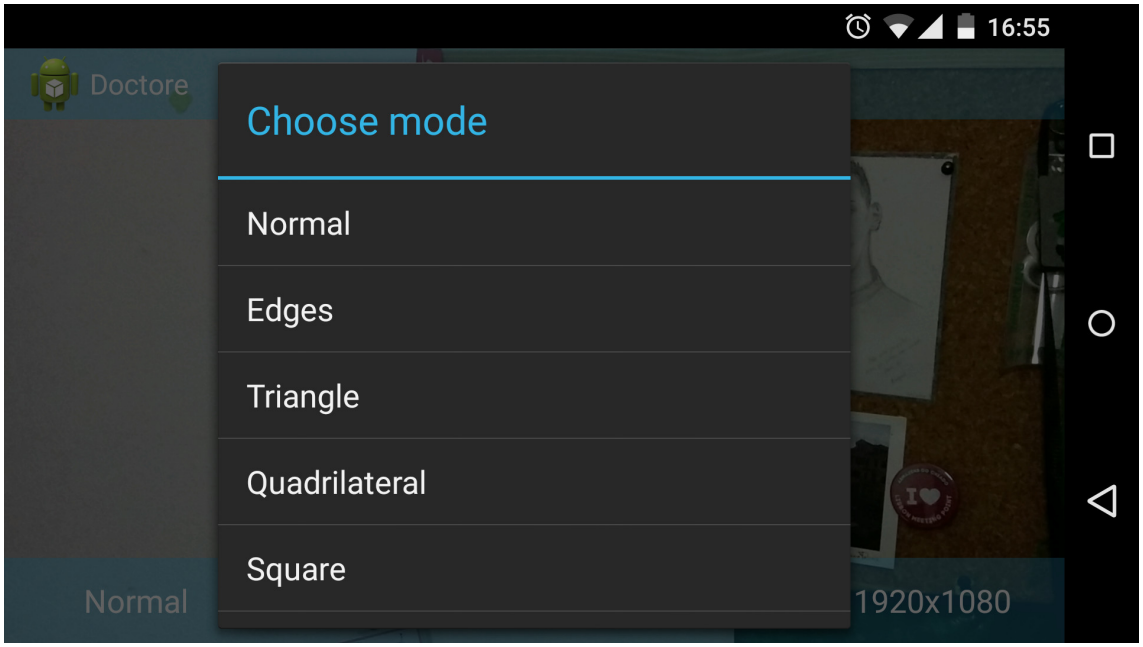


Figure 4.5: Mode selection

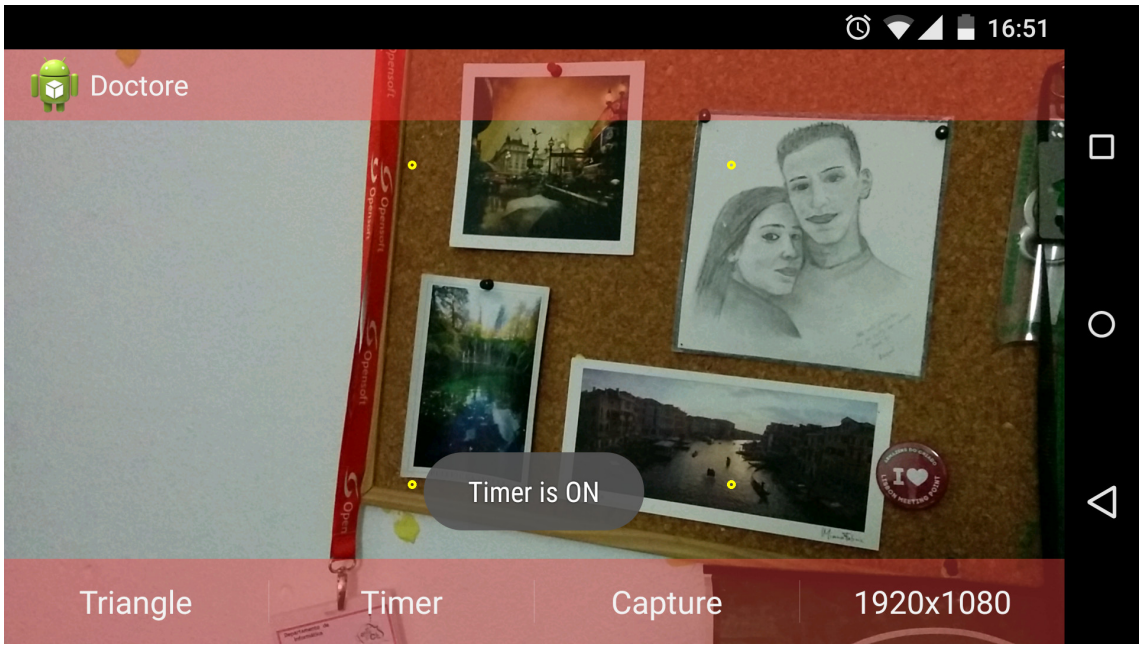


Figure 4.6: Triangle detection mode with timer activated

The next figures will show some examples of detections modes, plus the edges mode. Where applicable, two detections will be presented: one of a well defined shape and another of a more environment blended object.

Figure 4.7: Equilateral triangle detection

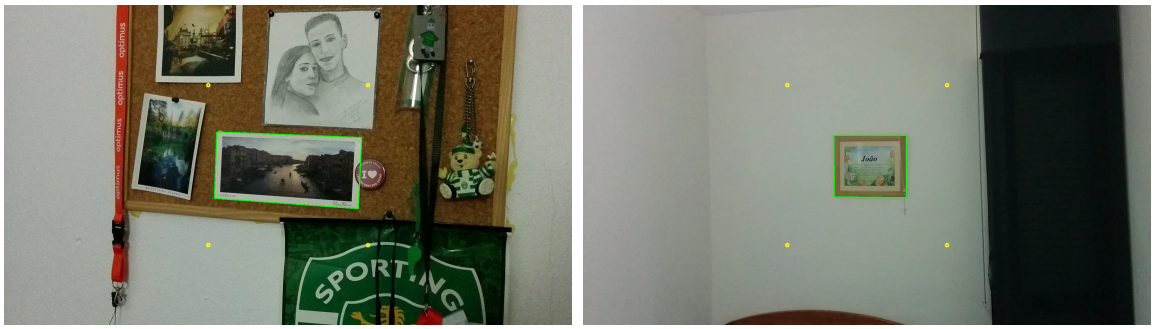


Figure 4.8: Quadrilateral detection (note on the left image how the approximation compensates the defect caused by the pin)

Figure 4.9: Square detection (note that the object in the left image is the same used in the exploratory application)

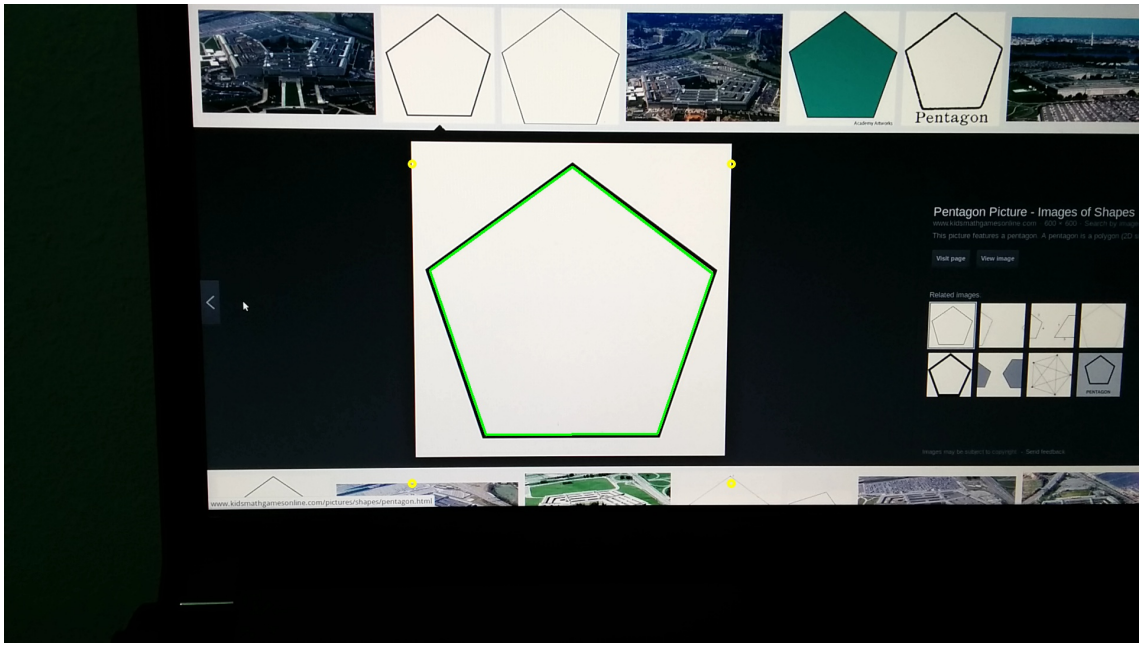


Figure 4.10: Pentagon detection

4.4 Discussion

While the prototype is functional and can be considered adequate for testing purposes, it is not as flexible as intended. Some actions, like changing the size of the ROI or the current edge extraction method cannot be made on the fly. Parameters cannot be tweaked dynamically either, requiring to delve back into the code for any change. This made small adjustments laborious, given the amount of configuration provided by the library. The interface has some minor visual bugs and lacks polish, given its prototype status. Nevertheless the system as a whole runs without any hindrance on several devices, allowing the performance tests on Chapter 5 to be conducted smoothly.

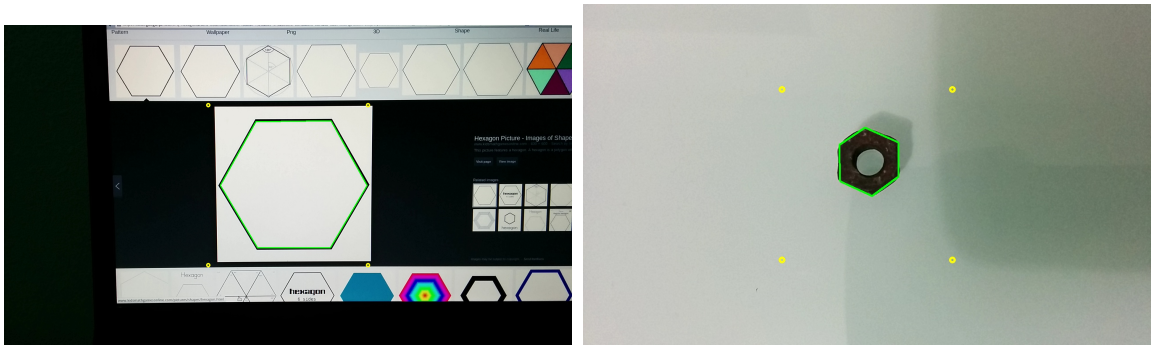
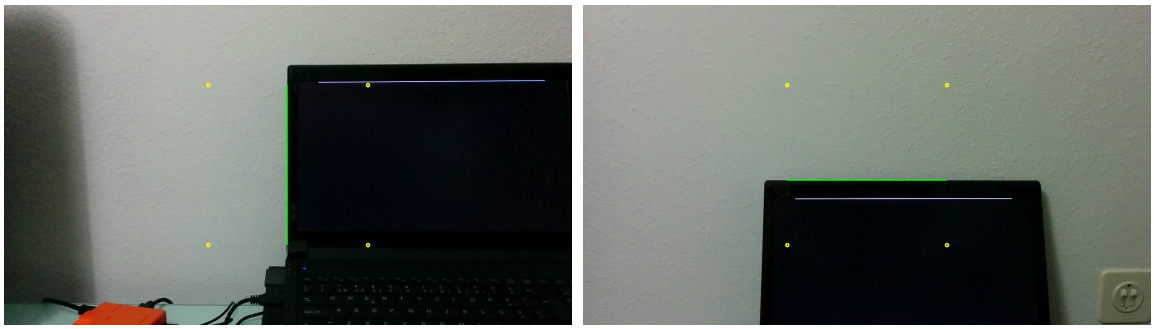


Figure 4.11: Hexagon detection

Figure 4.12: Circle detection



(a) Parallel line (b) Perpendicular line

Figure 4.13: Line detection (comparison angle is set to $\frac{\pi}{2}$)

Figure 4.14: Acute angle detection

Figure 4.15: Right angle detection

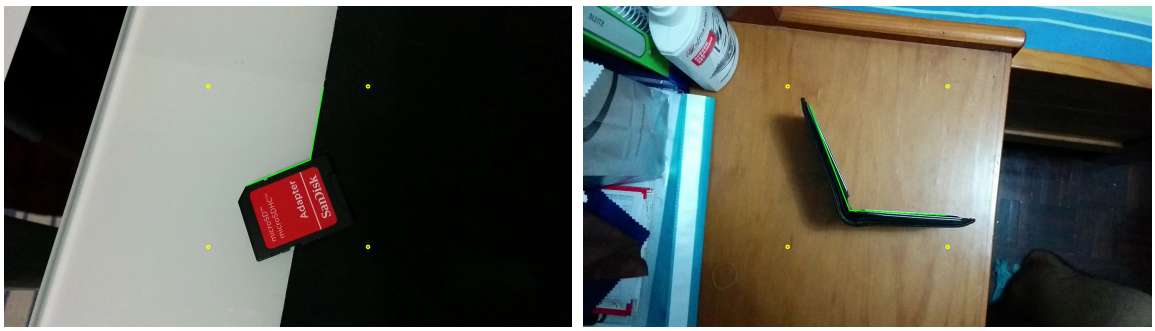


Figure 4.16: Obtuse angle detection

Figure 4.17: Symmetry detection (y-axis)

5

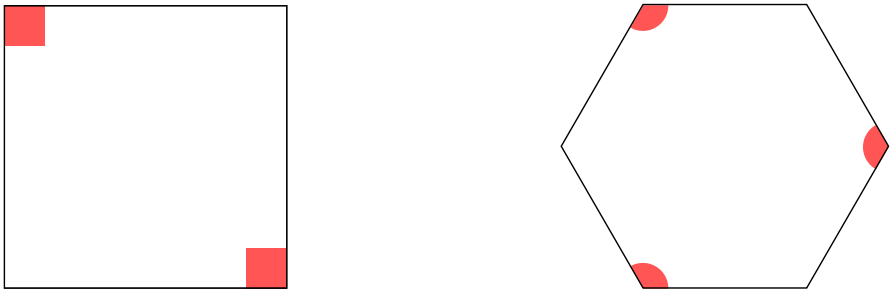
Evaluation

This chapter presents the evaluation of the library's performance, considering two main aspects: accuracy and speed. It is divided into four main sections. Section 5.1 provides insight into the key differences between the prototype used for performance testing and the prototype used in the user study. Section 5.2 details the testbed used for data gathering, while Sections 5.3 and 5.4 specify the actual testing and discuss the obtained results.

5.1 Testing Context

First and foremost, it is important to note that the library being tested is a relatively improved and optimized version of the one used in Chapter 6 as the user study was conducted rather early during the course of this dissertation – as soon as the first prototype was ready – which left a few months for tweaking, improvements and optimizations.

One such optimization comes with the realisation that it is unnecessary to check all angles in order to ascertain whether an approximation is a given regular geometric shape like an equilateral triangle, square, pentagon, hexagon, and so on. In fact, only as many as $\#vertices/2$ (rounded up to the nearest integer) are required, according to Figure 5.1.



(a) Square's checked angles (b) Hexagon's checked angles

Figure 5.1: Optimized angle checking for regular geometric shapes

Since all skipped angles are adjacent to, and share edges with, angles that have been measured, then by definition they must have the same measure as well. Other optimizations come from keeping object and variable copying to a minimum through heavy use of references and a stateful approach to JNI calls, for instance, where the Detector and the Painter do not have to be recreated on every call.

5.2 Testbed Setup

Two different testbeds were set up. One tested was comprised of a sheet of paper, henceforth referred to as "Paper" testbed, in which were printed several geometric shapes and uppercase letters, as can be seen in Figure 5.2 and consulted in Appendix A. It was devised with the intent of representing a best case testing scenario, under as many controlled conditions as possible. Such conditions entail a constant lighting source positioned to minimize shadows; highest possible contrast between shapes and background; opaque and low reflective surfaces; close inspection of perfectly depicted geometric shapes. The other testbed was comprised of mundane objects found in a room and strived to represent a real world testing scenario, and as such will be referred to as "Real" testbed. Apart from a constant lighting source, no other conditions were controlled. Unlike the Paper testbed however, these objects were found at different inspection distances, effectively testing the prototype in close, medium and long range detection. Figure 5.3 shows the objects tested upon.

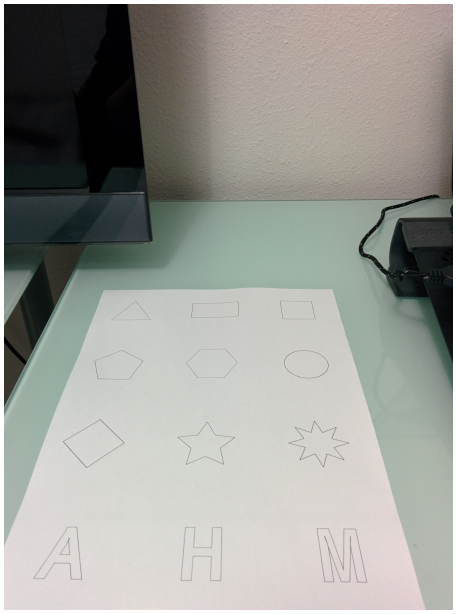
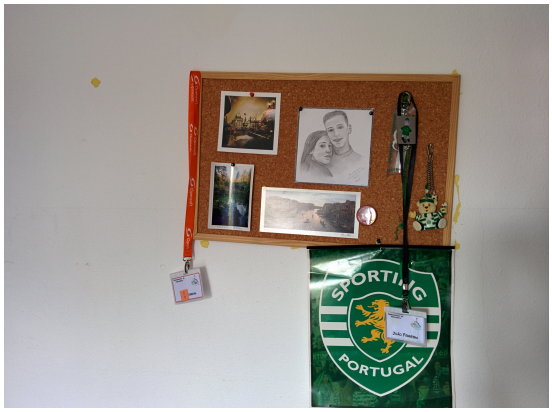


Figure 5.2: Best case scenario "Paper" testbed

(a) Very close range objects (≈ 5-10cm)

(b) Close range objects (≈ 20-50cm)



(c) Medium range objects (≈ 80-120cm)

(d) Long range objects (≈ 160-220cm)

Figure 5.3: "Real" world scenario testbed

5.3 Testing

Tests were conducted using two mobile devices:

LG Google Nexus 5 a high-end smartphone sporting a 5″ screen with a 1920x1080 (1080p) resolution, 2.3GHz quad-core processor, 2GB of RAM and a 13MP rear camera featuring Optical Image Stabilisation (OIS) technology.

Motorola Moto G (2013) a medium-spec smartphone sporting a 4.5″ screen with a 1280x720 (720p) resolution, a 1.2GHz quad-core processor, 1GB of RAM and a 5MP rear camera.

Both devices were running Android 4.4.4 KitKat and no particular care was taken in regards to whether WiFi and Bluetooth were on, or which OS applications were running in the background to simulate a normal usage.

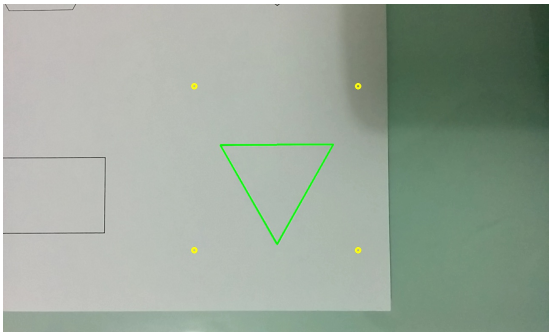
The library settings used for the tests were a 0.5 ROI ratio, based on the smallest frame side, and a 0.1 contour area-to-ROI ratio. As a title of example, for a resolution of 1920x1080, a centre-aligned 540x540 ROI was used. The 0.1 ratio means that any closed contour with an area smaller than ten percent of the ROI area would be immediately discarded. It only applies to shape and symmetry detection.

Testing involved detecting each geometric shape and notion five times while logging detection times for both success and failure cases. This logged time comprehends the moment when a frame is acquired from the camera to the moment when it is ready to display after being processed. The steps taken in any particular detection are as follows:

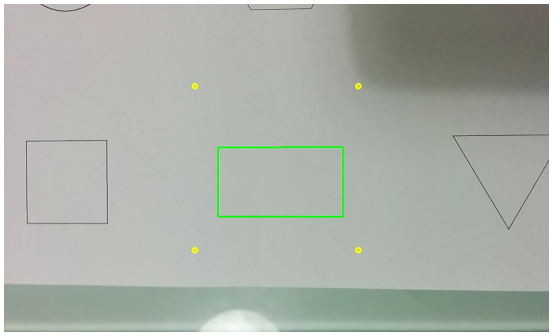
1. Switch to the relevant detection mode
2. Activate timer mode
3. Place the object within the ROI and halt all motion
4. Detection takes place
    - If detection is successful return to normal mode and deactivate timer
    - Detection continues until it succeeds, user moves the device or switches modes

Logging was done on every frame while detection was being attempted, despite the result. This allowed to measure how long a user would have to wait until the object shown to the camera was actually detected.

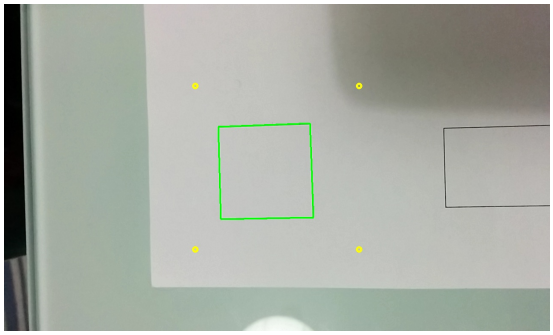
A total of six tests were performed overall, three per testbed: one using the Nexus 5 at a 1920x1080 native frame resolution; another using the Nexus 5 at a 1280x768 frame resolution; and the last using the Moto G at a 1280x720 native frame resolution. The reasoning behind the Nexus 5 non-native resolution test was to gauge performance gains by stepping down from FullHD to HD resolution. The same did not make sense for the Moto G since sub-HD resolutions incurred too much pixellation to strive for consistent results. It also allowed for a more close comparison between the two devices. Figures 5.4a through 5.5f show the detections on the "Paper" testbed and Figures 5.6a through 5.7f on the "Real" testbed.



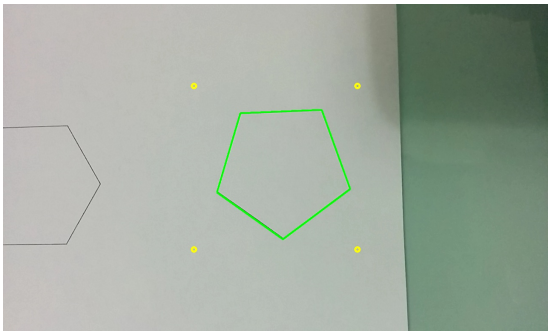
(a) Triangle



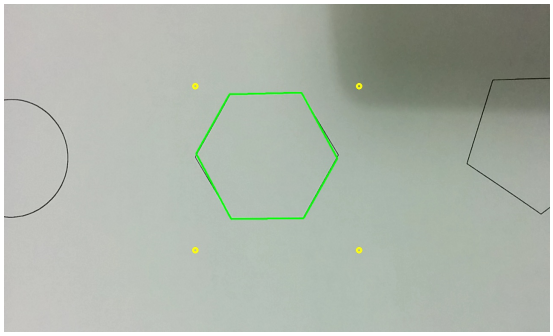
(b) Quadrilateral



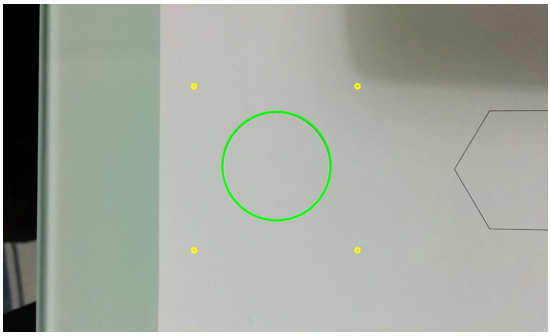
(c) Square



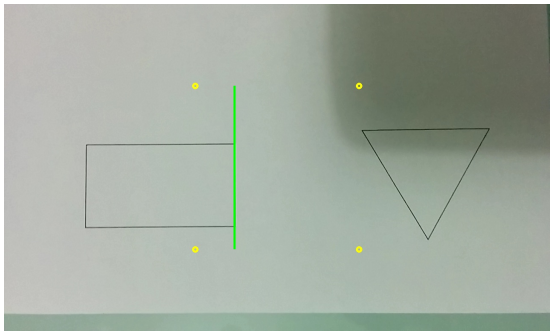
(d) Pentagon



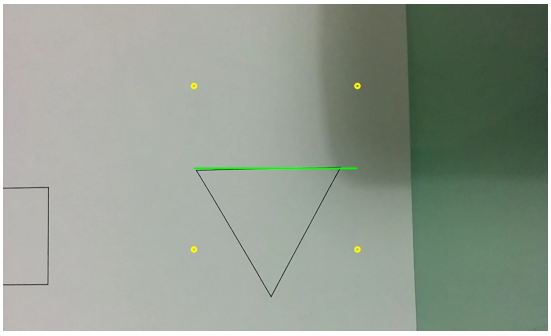
(e) Hexagon



(f) Circle

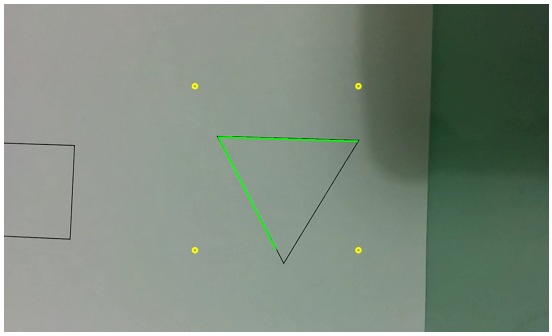


(g) Parallel line (to the y-axis)

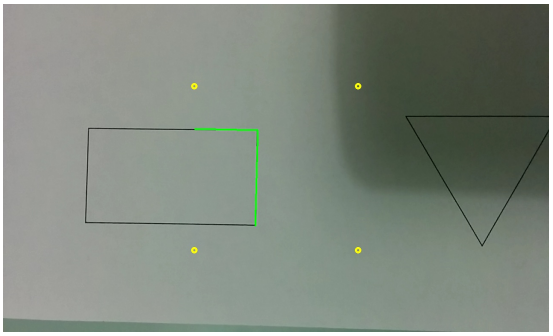


(h) Perpendicular line (to the y-axis)

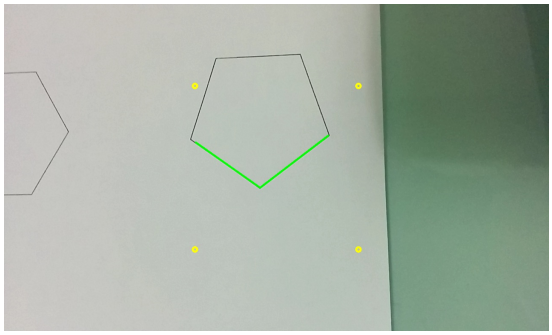
Figure 5.4: Detection on “Paper” testbed (1)



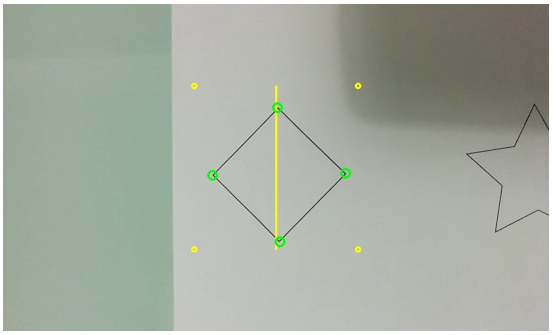
(a) Acute angle



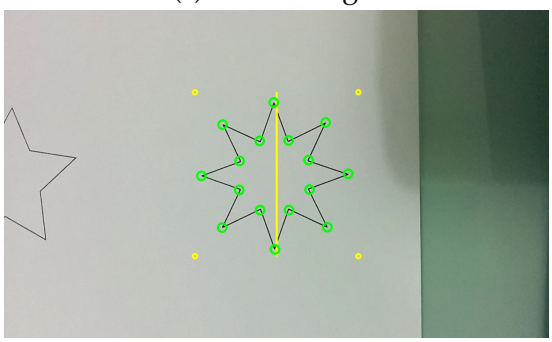
(b) Right angle



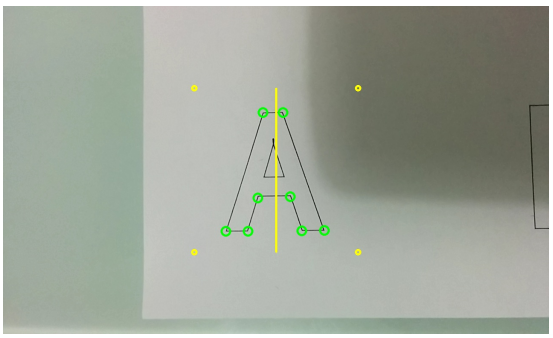
(c) Obtuse angle



(d) Rhombus



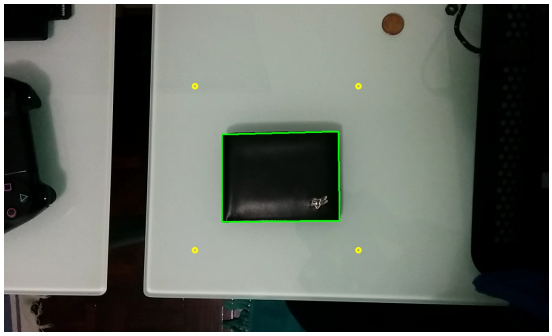
(e) Compass rose



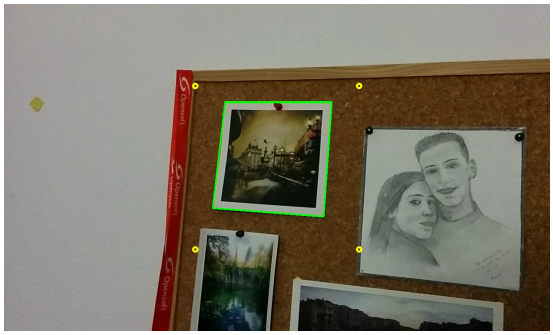
(f) Letter 'A'

Figure 5.5: Detection on "Paper" testbed (2)

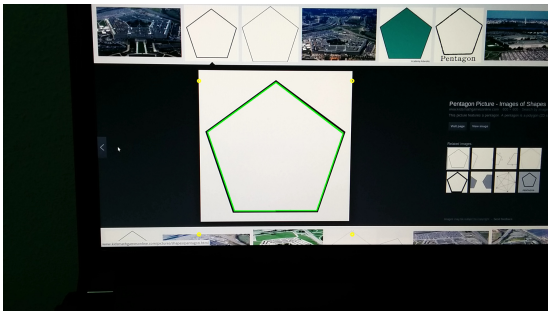
(a) Triangle



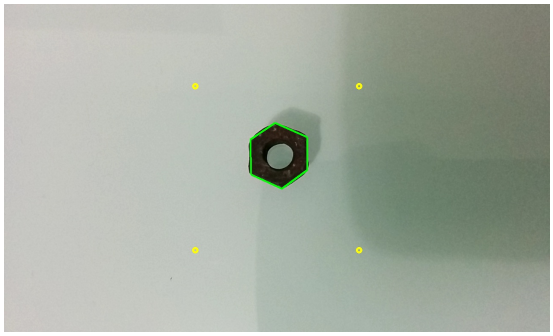
(b) Quadrilateral



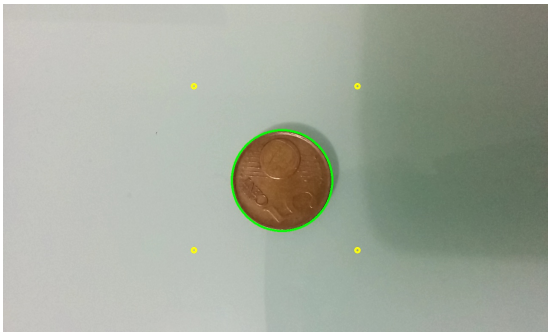
(c) Square



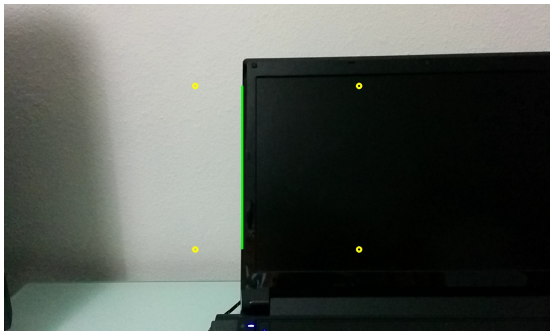
(d) Pentagon



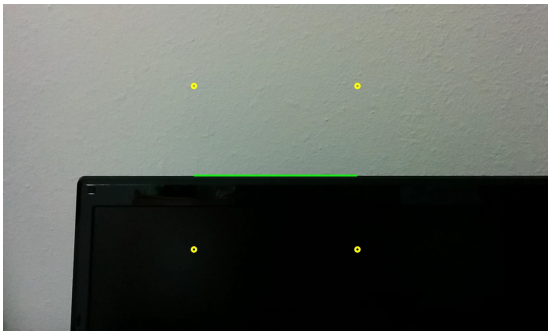
(e) Hexagon



(f) Circle

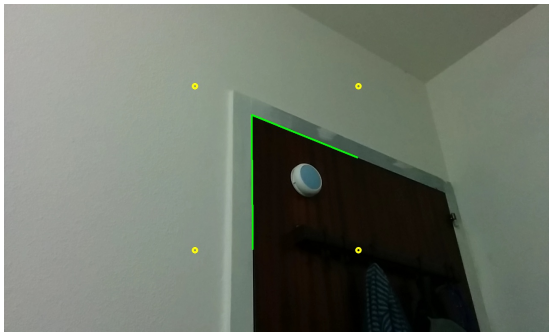


(g) Parallel line (to the y-axis)



(h) Perpendicular line (to the y-axis)

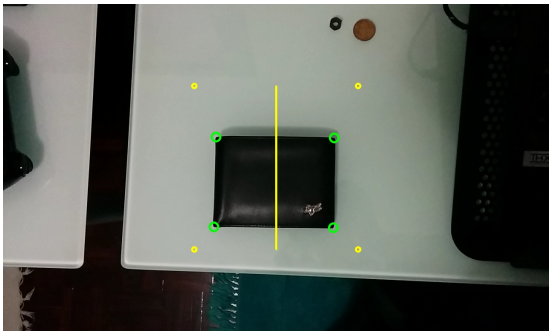
Figure 5.6: Detection on “Real” testbed (1)



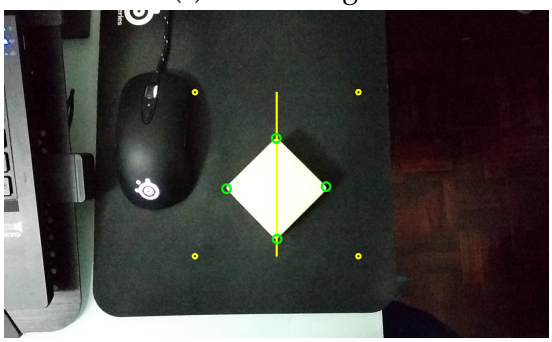
(a) Acute angle

(b) Right angle

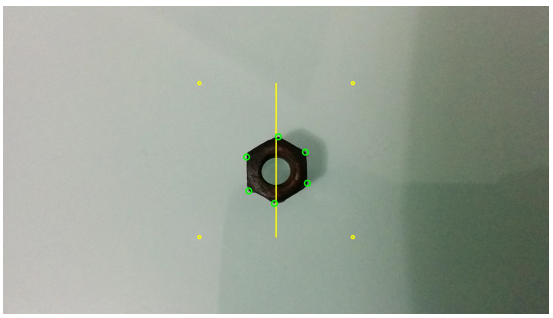
(c) Obtuse angle



(d) Wallet



(e) Box



(f) Screw

Figure 5.7: Detection on “Real” testbed (2)

5.4 Results

The primary performance goal was to keep the highest possible frame rate to ensure a smooth real-time experience. On most mobile devices this means keeping up with a frame rate output of thirty frames per second. This holds true for both the Moto G and the Nexus 5. Therefore, and as mentioned above, frame rate performance is measured in terms of frame time: how long it takes to process a newly received frame and display it. Equation 5.1 indicates the amount of time available for processing a single. Thus, for a frame rate of thirty frames per second the frame time available is approximately 33 milliseconds. Failure to process and deliver a frame within this period of time will result in incoming frames having to wait to be processed, essentially culminating in frame rate loss, perceived by the user as stuttering/lagging.

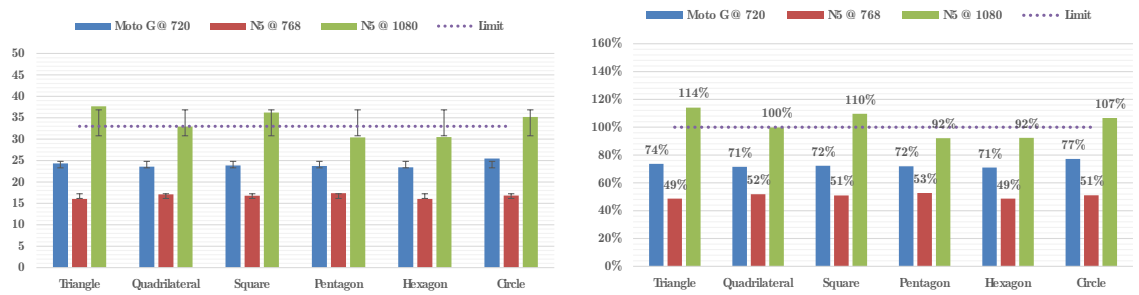
$$FrameTime = \frac{1\ second}{FrameRate} \tag{5.1}$$

A secondary performance goal is related to the quality of the experience, which means reaching a high frame rate at the mobile device's camera native resolution. While not as critical as sustaining a high frame rate, it can improve the experience in the eyes of the user who might be expecting to use her device in a similar way as when recording a video: without being restricted to lower, non-native resolutions.

Prior to testing, the prevalent assumption was that the Nexus 5 would struggle with keeping the thirty frames per second at the demanding native 1920x1080 resolution, despite its high processing power within the smartphone environment. However, the most prominent question was not only how much differently it would perform at a non-native resolution, but also how it would fare against the midrange Moto G, in order to ascertain whether the dominant differential factor in performance would be due to processing power and RAM availability or resolution. Another pertinent question was related to range: how easy would it be for the prototype to perceive the same object at different ranges, specifically if long range objects, whose defects are somewhat concealed by the distance, can be more easily identified. The last question these tests sought to answer was how well transparent and reflective objects can be detected. All graphs below contain upright lines representing the standard deviation, where applicable.

5.4.1 Paper Testbed

Figure 5.8 shows that regular geometric shapes can be detected as fast as half of the available frame time, and even less in some cases. The Moto G was capable to detect all shapes comfortably within the limit at its native 1280x720 resolution. The Nexus 5 managed to outperform the Moto G at the slightly higher 1280x768 resolution. However it struggled to keep detection times under the limit when operating at its native 1920x1080 resolution. Interestingly enough, the worst result came from triangle detection, which was expected to be one of the fastest detectable shapes, given its algorithmic simplicity. Although not as discrepant, it was also the case with the Moto G. This could be due to inadvertently adding motion blur to the frame, consequently causing a longer run of the edge extraction step. Nevertheless, even in this worst case the frame rate loss was minimal since the next frame would be delayed only a mere five milliseconds, hardly perceived by the human eye.

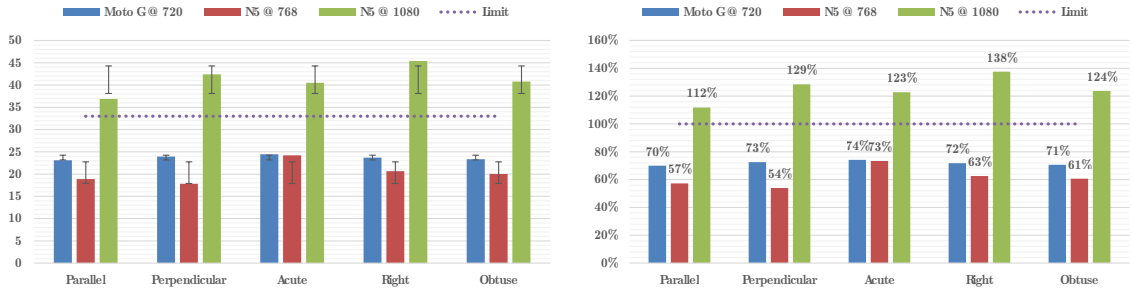


(a) Mean frame time (ms) across all devices (b) Mean %time spent processing one frame

Figure 5.8: Geometric shapes results on "Paper" testbed

On Figure 5.9 are presented the results concerning geometric notion detection. It is further accentuated the struggle of the Nexus 5 running at native resolution, taking nearly forty percent longer past the limit in the worst case. The resulting frame rate loss remained imperceptible however. Detecting geometric notions was also slightly more demanding of the device when running at non-native resolution, bringing it more in line with Moto G's performance. Still both devices managed to perform well under the frame time limit.

Unlike the other detection techniques above, where the target object is always the same according to its geometric concept, symmetry detection is increasingly demanding the more complex the symmetric object is. Figure 5.10 demonstrates that all devices had to put more effort into detecting the three symmetric shapes.



(a) Mean frame time (ms) across all devices (b) Mean %time spent processing one frame

Figure 5.9: Geometric notions results on "Paper" testbed

That being said, the Nexus 5 showed an improvement running at native resolution, managing to perform within the frame time limit when detecting the rhombus. The compass rose expectedly proved the most difficult, with the Moto G coming rather close to the limit.

(a) Mean frame time (ms) across all devices (b) Mean %time spent processing one frame

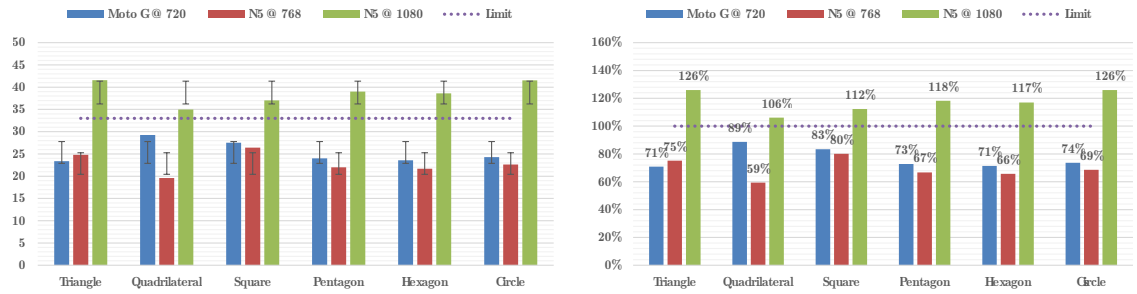
Figure 5.10: Symmetry results on "Paper" testbed

Probably the most noteworthy achievement of this test was that all detections were successfully performed on the first frame received once detection was triggered. Thus, under controlled conditions this prototype has a very low detection wait time. Although not being able to meet the target frame rate at higher resolutions, frame rate loss was also deemed minimal and non-detrimental.

5.4.2 Real Testbed

As mentioned previously, the "Real" testbed had very few controlled conditions, therefore results shown below provide a better representation of what to expect in a live, production environment. Figure 5.11 promptly demonstrates an increased frame time across all devices. Despite this, both the Moto G and the Nexus 5 running at non-native resolution manage to stay below the frame time limit. Given

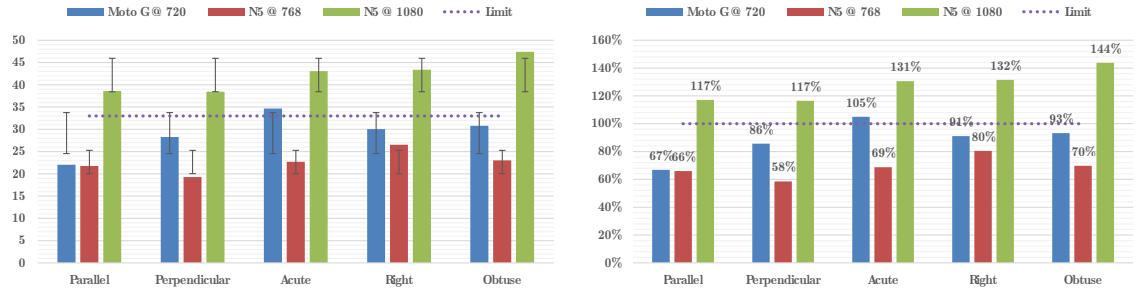
that the Nexus 5 struggled when running at native resolution during tests involving perfect, defect-free depictions of geometric shapes, it stands to reason that such struggle would escalate on objects which themselves fell short of ideal, either for not being entirely regular, not having perfectly delineated edges, being somewhat reflective or simply not having enough contrast with their current background.



(a) Mean frame time (ms) across all devices (b) Mean %time spent processing one frame

Figure 5.11: Geometric shapes results on "Real" testbed

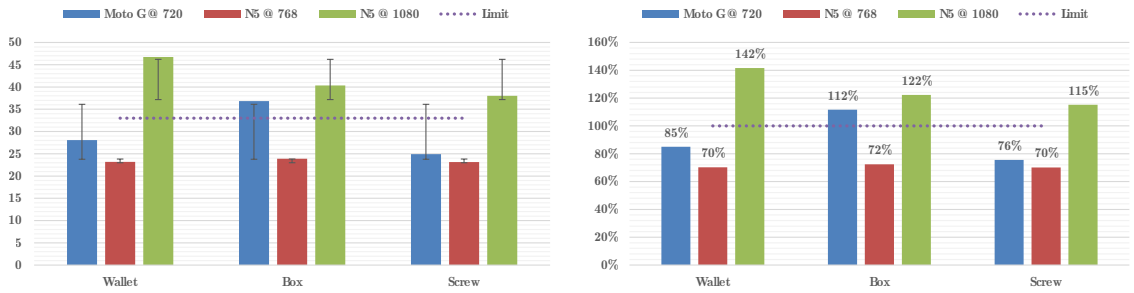
Detecting geometric notions finally took its toll on the Moto G, as Figure 5.12 shows. It came close to the frame time limit on most geometric notions and finally went over it during acute angle detection, although by a mere two milliseconds. The Nexus 5 running at non-native also saw an increase in frame time, but still scored comfortably within the limit. Incidentally it also achieved the worst result during obtuse angle detection when running at native resolution, taking up nearly an extra half frame worth of processing.



(a) Mean frame time (ms) across all devices (b) Mean %time spent processing one frame

Figure 5.12: Geometric notions results on "Real" testbed

The symmetry detection test also saw the Moto G go over the limit. Unsurprisingly, the Nexus 5 also went over the limit at native resolution, albeit performing better than expected at non-native resolution. Figure 5.13 displays the results.

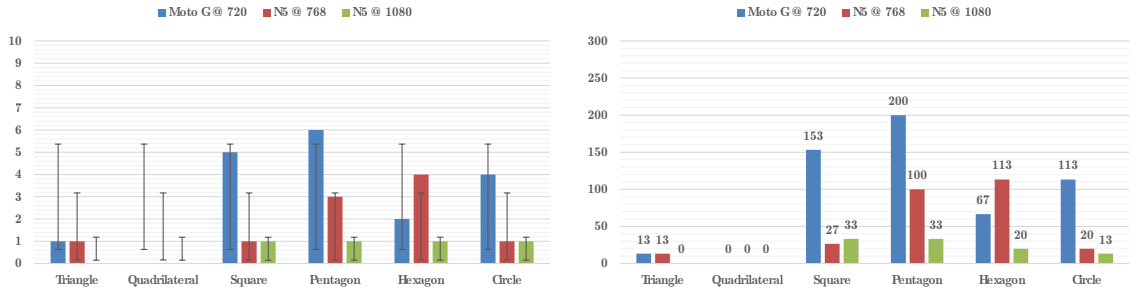


(a) Mean frame time (ms) across all devices (b) Mean %time spent processing one frame

Figure 5.13: Symmetry results on "Real" testbed

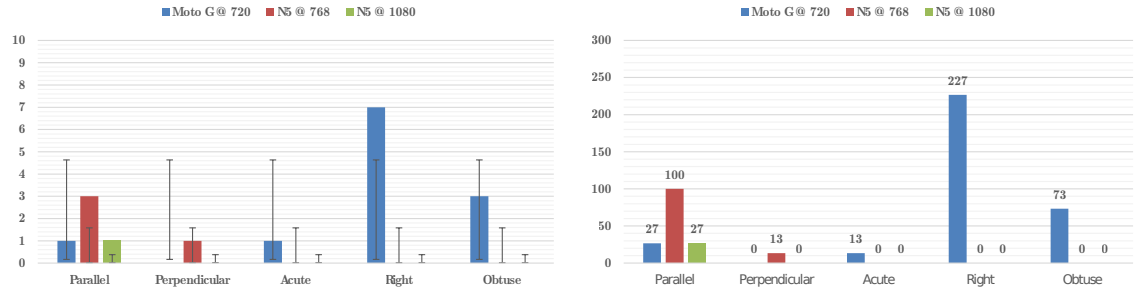
Unlike the "Paper" testbed, these tests weren't flawless. Several shapes and notions weren't successfully detected upon the first frame processed. Therefore an assessment of the time spent, as well as the number of frames processed until successful detection was required in order to provide a better insight into how much that would translate into effectively wasted time, in itself a form of accuracy measurement.

Figures 5.14 through 5.16 show an interesting trend: despite going over the frame time limit most of the time during a successful detection, the Nexus 5 running at native resolution actually achieves the lowest amount of frames-to-success (FTS) and consequently the fastest time-to-success (TTS). On the other hand the Moto G managed to take nearly a quarter of a second until a successful detection was accomplished, whereas the Nexus 5 running at non-native resolution found the middle ground.



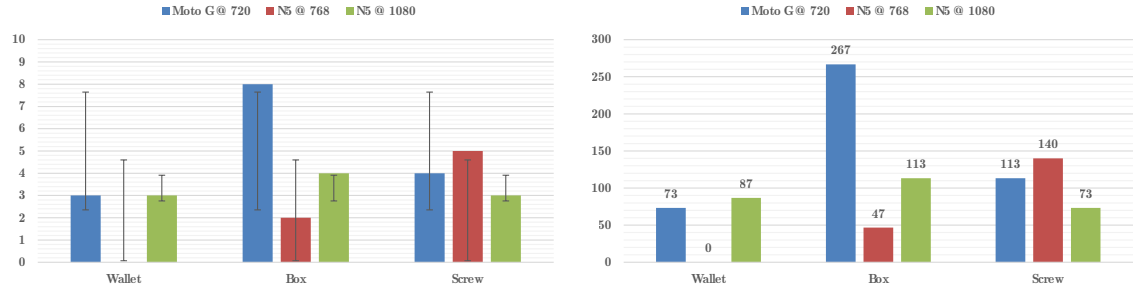
(a) Mean FTS (b) Mean TTS (ms)

Figure 5.14: Geometric shapes: unsuccessful results



(a) Mean FTS (b) Mean TTS (ms)

Figure 5.15: Geometric notions: unsuccessful results



(a) Mean FTS (b) Mean TTS (ms)

Figure 5.16: Symmetry: unsuccessful results

While it is important that successful detection happens within the frame time limit, it is equally important to come to the conclusion that a frame either holds no meaningful features, or the shape/notion is not the one sought after, and therefore must also be disregarded within said limit, in order to achieve the target frame rate. In light of this, Figures 5.17 - 5.19 show the mean time elapsed to disregard such uninteresting frames. Note that the absence of time in some graphs is in accordance with the fact that those specific shapes or notions had a flawless detection record, meaning a successful detection was made upon the first processed frame.

As can be verified, all devices managed to stay practically under the frame time limit. The only exception was the Nexus 5 running at native resolution, but even then it was for a single millisecond.

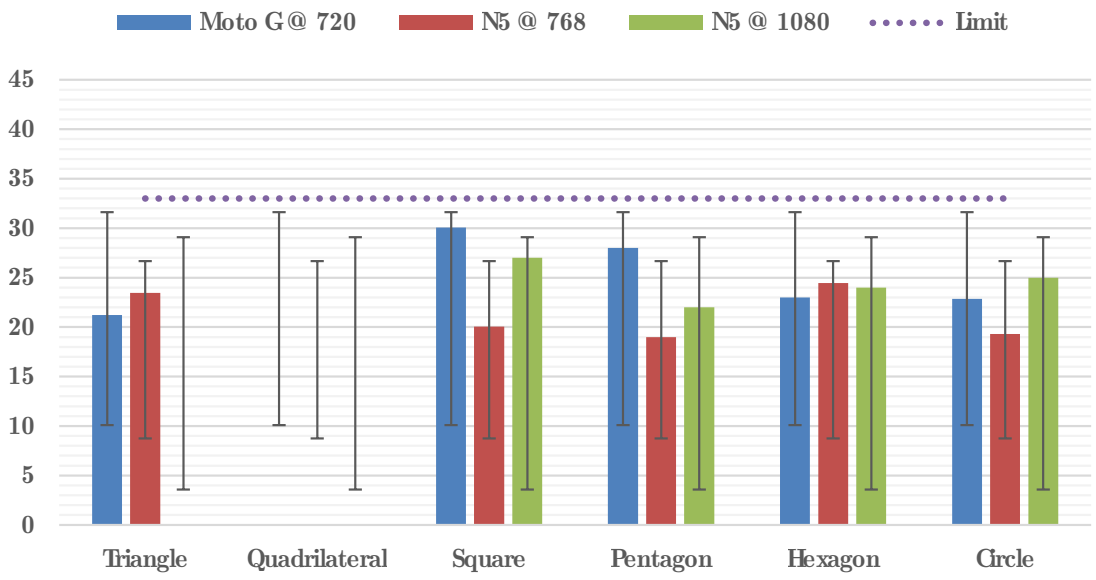


Figure 5.17: Mean frame time (ms) to disregard an uninteresting shape

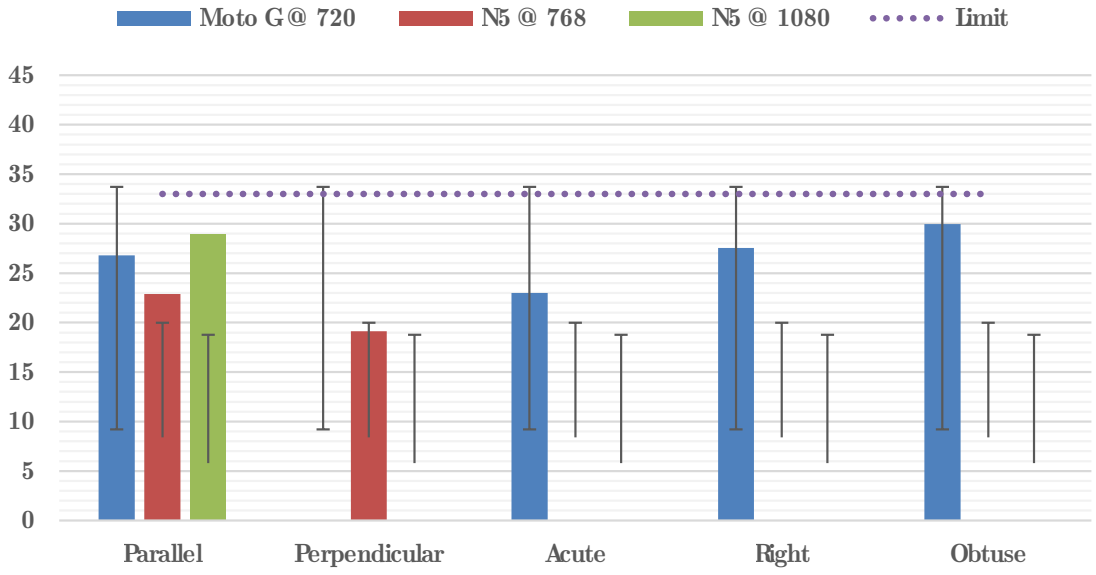


Figure 5.18: Mean frame time (ms) to disregard an uninteresting notion

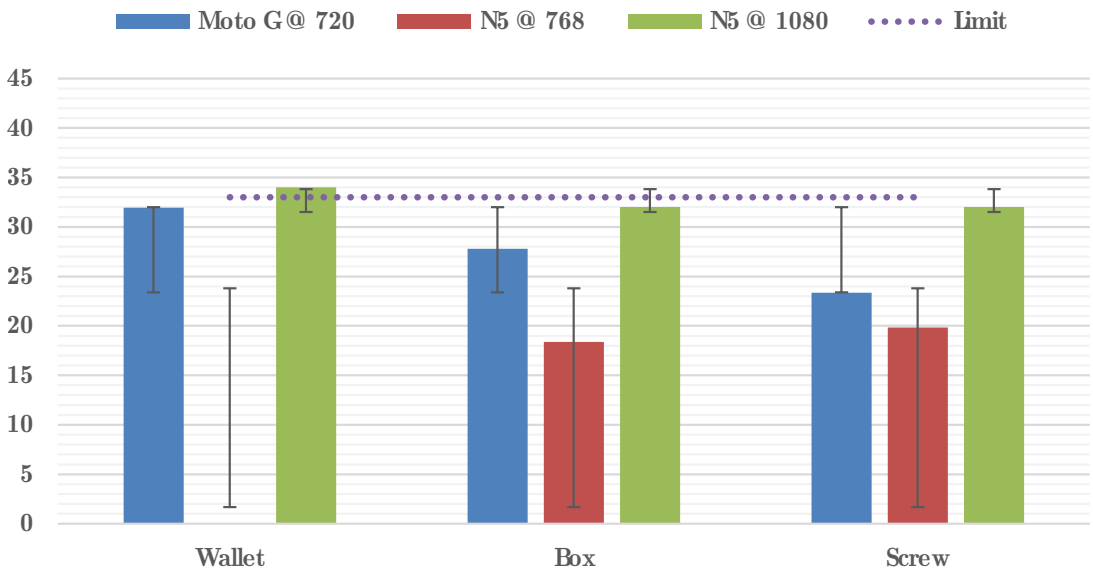


Figure 5.19: Mean frame time (ms) to disregard an uninteresting symmetry

5.4.3 "Paper" vs "Real"

To bring the picture into full view, a comparison of the various results displayed above was made between the same device on both testbeds. The rhombus object in the Figures below refers to the "Box" object used previously, as it was a square box rotated to act like a rhombus, which was also present on the paper sheet and provided a comparison point. Since the other objects used for symmetry detection in the "Real" testbed do not match the ones used for the same purpose on the "Paper" testbed and thus cannot be compared, they were omitted. The Moto G performed without a significant difference on both testbeds, around twenty-five milliseconds, until halfway through the geometric notions (Figure 5.20). The Nexus 5 running at non-native resolutions had an increase in frame time on the second testbed, but still lingered around the twenty-five millisecond mark, outperforming the Moto G on almost all cases (Figure 5.21). Finally, when switching to its native resolution, the Nexus 5 only managed to stay under the frame time limit in four occasions, all belonging to the best case scenario testbed (Figure 5.22).

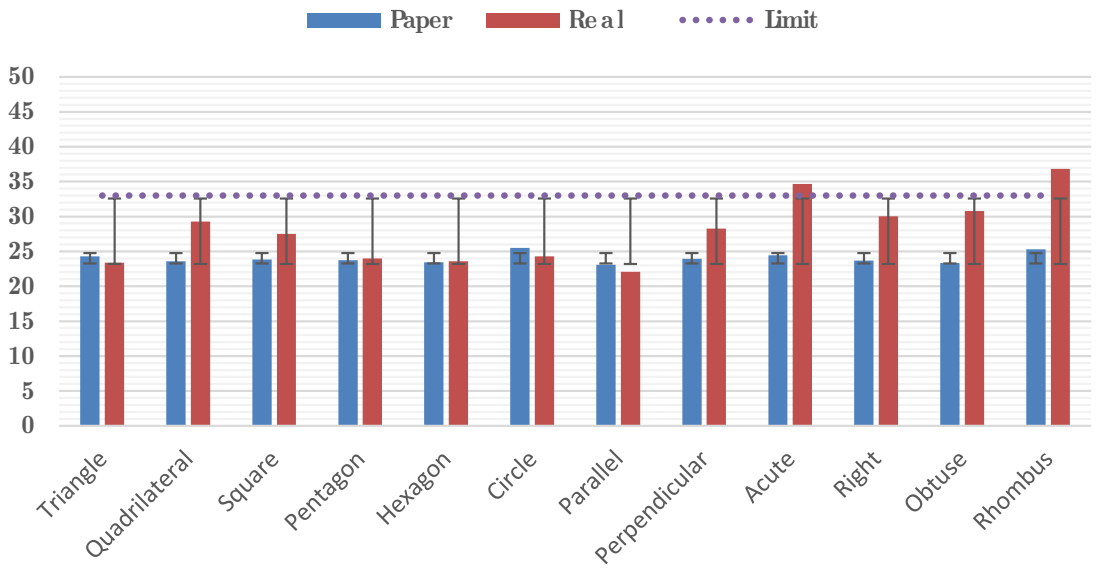


Figure 5.20: Moto G @ 720 mean frame time (ms) comparison

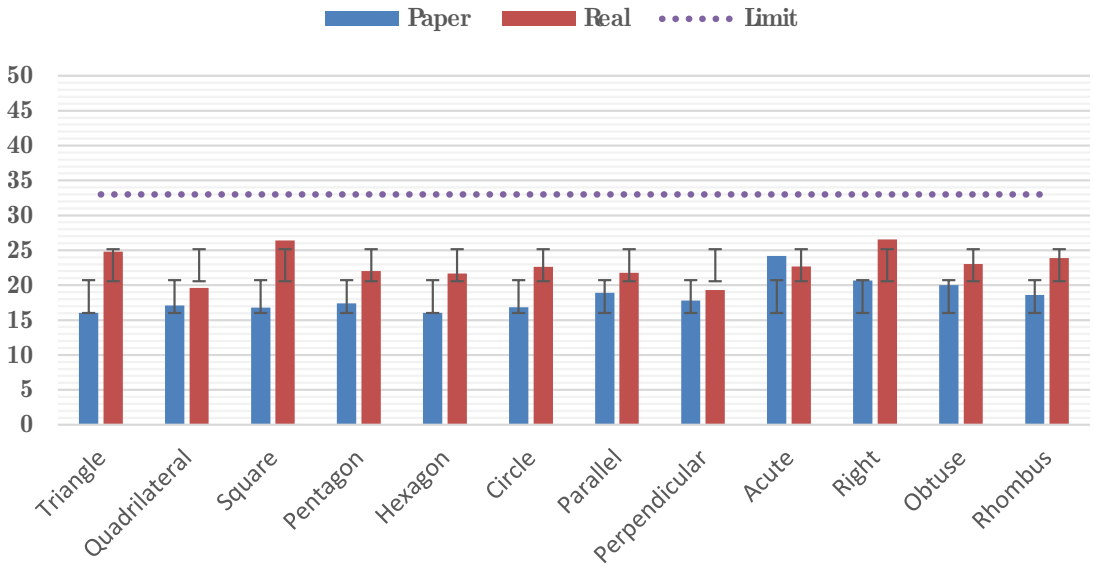


Figure 5.21: Nexus 5 @ 768 mean frame time (ms) comparison

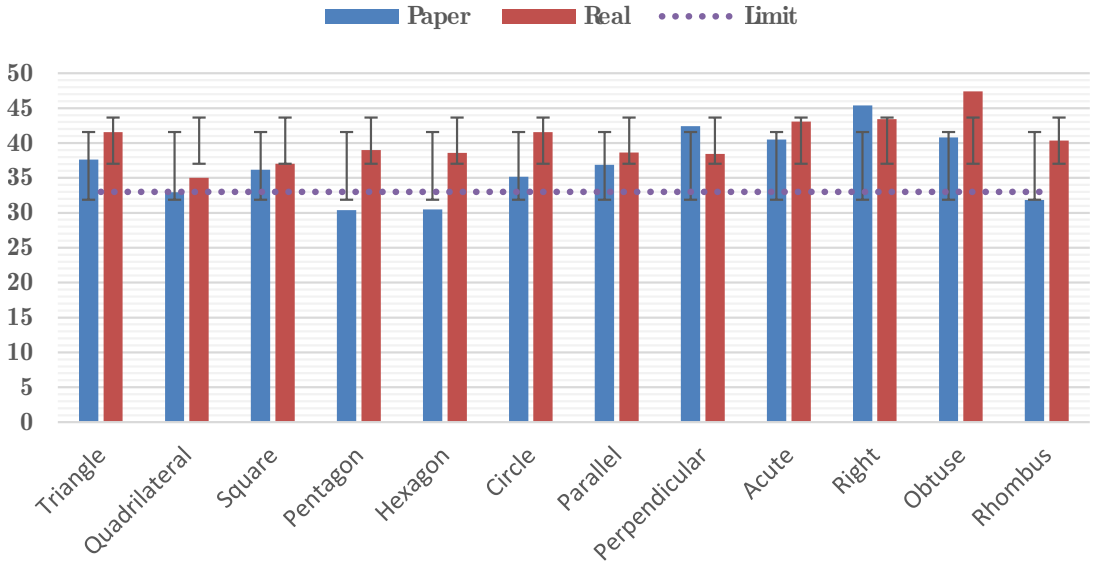


Figure 5.22: Nexus 5 @ 1080 mean frame time (ms) comparison

5.4.4 Issues

Given the exploratory concept of this prototype, a multitude of objects can be used to infer geometric shapes and notions. However some objects pose a challenge to image processing due to their characteristics. Transparent and reflective objects, such as the ones shown in Figures 5.23 - 5.28 proven difficult to detect properly. The issue resided with the edge extractor (Section 3.3.1.1). Throughout the tests above, the Canny operator was the chosen approach for edge extraction due to its ability to produce clear edges, avoid false positives and reduce leftover noise. This would substantially simplify and accelerate the following contour approximation stage. On the other hand the adaptive threshold would leave a high amount of leftover noise, requiring further cleaning to achieve the same effect. When dealing with transparent and reflective objects however, the Canny operator struggled to mark the edges closer to the transparent or reflective portion of the object. Since the adaptive thresholding technique deals better with illumination gradients, it was switched in and a comparison of both methods can be seen in the Figures below.

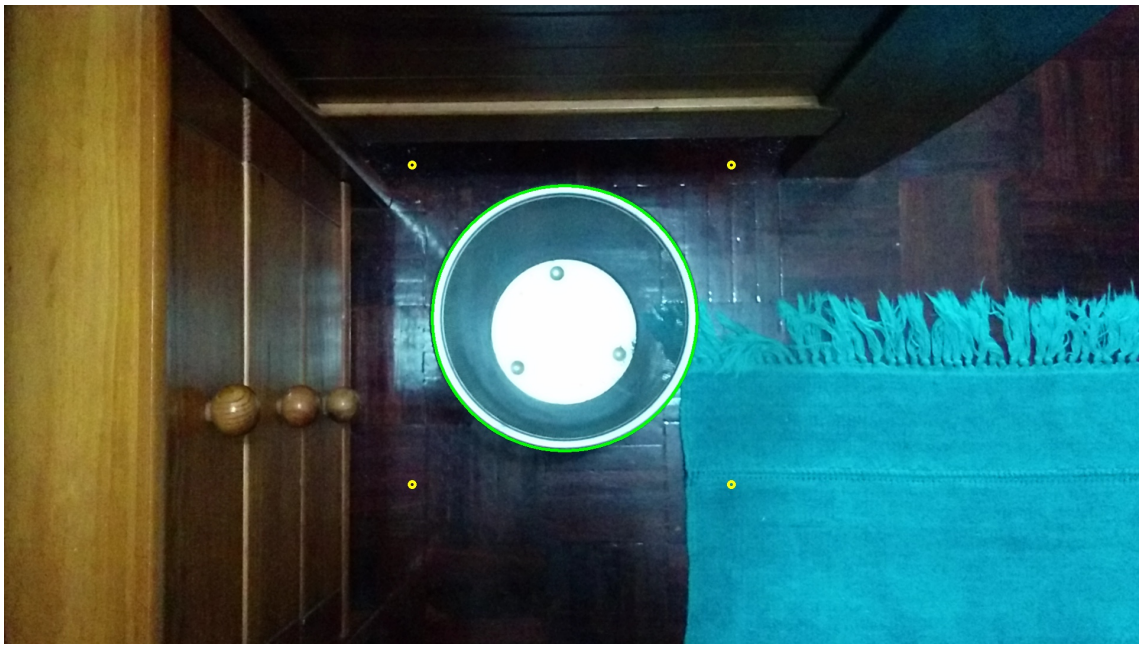
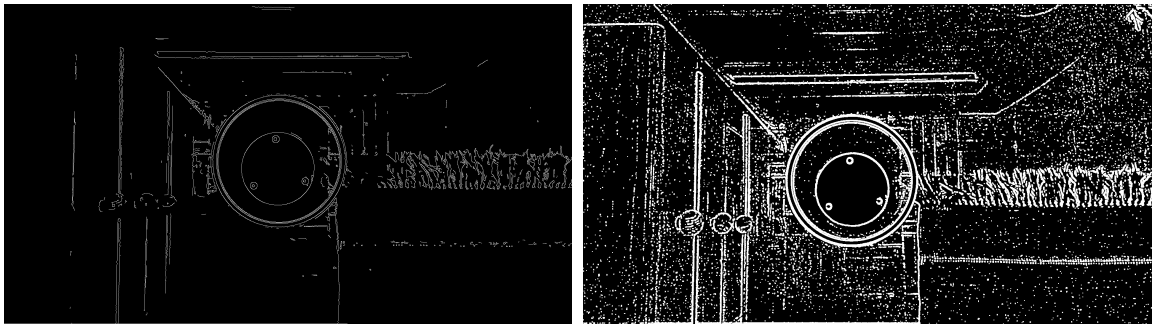


Figure 5.23: Reflective garbage bin



(a) Canny Operator (b) Adaptive threshold

Figure 5.24: Bin edges comparison

Figure 5.25: Reflective cube

(a) Canny Operator (b) Adaptive threshold

Figure 5.26: Cube edges comparison

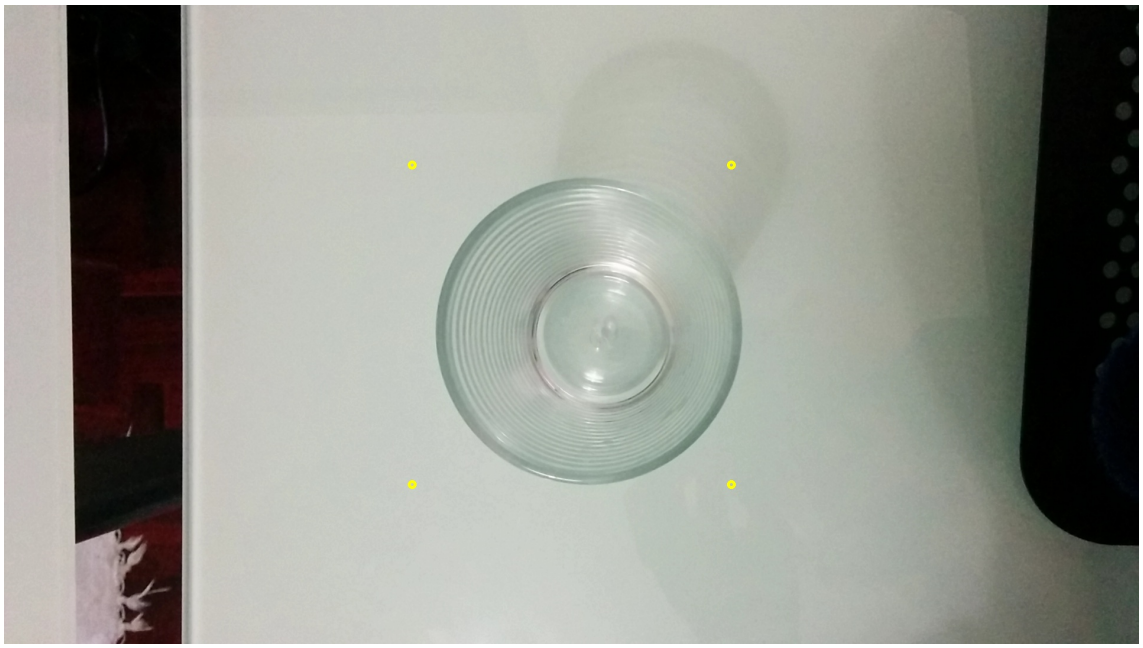
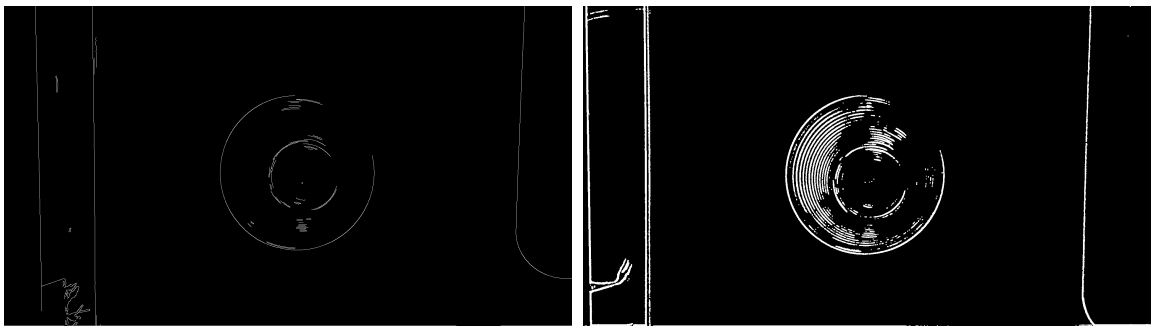


Figure 5.27: Transparent glass cup



(a) Canny Operator (b) Adaptive threshold

Figure 5.28: Glass cup edges comparison

Both methods were able to delineate the bin against the dark background, and thus it (the circle) was detected (Figure 5.23). When dealing with the reflective cube on a less contrasting background, the Canny operator could not identify the top edge, whereas the adaptive threshold managed to delineate it. However, even then the square would not be detected. One possible reason was the inability of the adaptive threshold to separate the top line from the square and the object as a whole – square plus line – was checked for contours in the later stages of the detection. Finally a portion of the transparent glass cup edges were not detected by either method, making detection impossible.

5.4.5 Discussion

First and foremost, the results show that the target frame rate can be met for simple geometric shape and notion detection. They also confirm the assumption that the Nexus 5 would struggle running at its native resolution of 1920x1080, hence answering the first question: resolution is indeed the largest factor in differencing performance, rather than better or worse specifications. Of course a combination of both factors can tip the scales towards either end of the performance spectrum: this is why the Nexus 5 running at a non-native resolution outperformed the Moto G at roughly the same resolution and was the only device to always stay under the frame time limit. However the performance penalty was noticeable when increasing the resolution, which raises the question of how will the detection system cope with newer smartphones and tablets already featuring resolutions beyond FullHD but not the processing power to match. Despite this, the Moto G had a more than acceptable performance, which indicates the detection system performs well on a wide range of devices.

Concerning the fact that some shapes and notions, namely angles and line parallelism, took longer to be detected than more complex notions such as symmetry, despite being simpler, stems from the fact that, for instance, when detecting for an acute angle in a zig-zag line, each vertex has the potential to conform to the type of angle sought after, which means not only that more contours are analysed but the same contours are analysed from one end to the other before being discarded. On the other hand shapes and complex symmetries have more strict concepts and therefore a bad contour can be discarded halfway it's analysis, thus speeding up processing.

Other factors like the operating system, the data logging (which was done on every frame during detection) and hardware features can also impact performance. Recall that the Nexus 5 features OIS, a mechanism designed to stabilise

recorded imagery, which might be the reason why it had better frames-to-success and time-to-success ratios over the Moto G. The latter seemed to take longer to focus and was rather susceptible to small jerking, blur inducing, movements, requiring a steadier hand. The major issue of the detection system was dealing reflective and transparent objects. This could be improved with more processing at this edge extraction level. For instance an RGB frame could be used instead of a greyscale frame and have its three respective channels processed. However, it would further increase the frame time limit, risking an unstable frame rate. In the end, whenever the limit was exceeded it was by a small margin, which is hardly perceivable by the human eye.

6

User Study

In this chapter is described how the user study was conducted and the experiment results are presented along with the user feedback obtained. Section 6.1 provide the context in which the study was made, whereas Section 6.2 reports how exactly the experiment was carried out. Afterwards, Section 6.3 analyses the choices and behaviour of the participants throughout the tests and Section 6.4 interprets the results and feedback obtained from the gathered data.

6.1 Study Context

In order to assess the educational factor and obtain direct user feedback, an experiment was undertaken in a elementary school, Escola Básica nº 3 of Almada. It ran across all school years except the first in descending order. The reason for excluding the first year will be discussed in Section 6.2. A total of 30 students (7 - 10 years old, average 9 years old) were chosen to participate: 3 classes (1 per year) $\times$ 5 students $\times$ 2 genders (10 per class). The teachers were asked to select the two students at the opposite ends of the prominence spectrum and the other eight randomly, while respecting gender equality. The participants came one at a time to the teachers' room where the experiment took place, which can be seen in Figure 6.1.

This room was not altered in any way in order to stay as close as possible to what one would expect to find in a elementary school: a room full of books,

Figure 6.1: The experiment room

instruments and other educational objects used by the teachers in their lessons, along with day-to-day objects such as cabinets, computers, televisions and coffee machines.

6.2 Study Description

The participants would come inside where they were explained what the experiment consisted of, what was required of them and then perform the experiment. After a brief introduction, they were told that the assigned smartphone, an LG Google Nexus 5, could detect most of the shapes and geometric notions they had been learning in class through its camera and that they were going to play a simple game using this "capacity" – the device would ask them to show it a particular geometric shape or geometric notion, and they would have 60 seconds to do it, otherwise it would move on to the next question, for a total of 5 questions. The participants were then free to walk around the room while they attempted to answer the questions. No further input or help was granted and each run took at most 5 minutes.

The questions were tailored to match the relative knowledge of each year. Less advanced years had clear, direct questions (*i.e.* show a triangle) and fewer questions pertaining to geometric notions, such as line parallelism. More advanced years not only had more questions related to geometric notions such as angles,

but geometric shape questions were asked in a indirect way (*i.e.* a geometric shape without vertices instead of a circle). Therefore a total of 20 questions (4 classes $\times$ 5 questions) were elaborated, and their order would be randomised on every run.

The first year was excluded once it was determined that a few second year participants were struggling to read some questions despite their simple wording, and it was agreed upon with the teachers that the first year participants would struggle even more, to the point of hindering the experiment.

Each participant filled out a post-hoc questionnaire (written in Portuguese, the participants native language), designed to assess the difficulty, utility and fun factor of the experiment, and also the acquaintance level with mobile devices using Likert-scale questions ranging between 1 and 5 (1 = worst, 5 = best) and a space for comments and observations.

Total time for study was approximately five hours split over two days. This includes the acclimatising introduction and questionnaire filling.

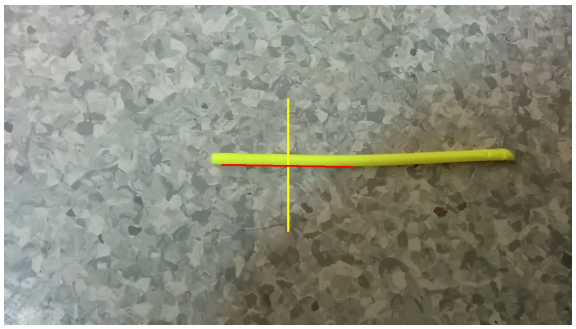
6.3 Observational Analysis

During the course of the experiment, the participants have shown to understand clearly what they were supposed to accomplish, and that they feel comfortable using mobile devices at such young ages. Most issues that arose were related to their knowledge (or lack thereof) regarding the subject, with a small portion pertaining to the handling of the device itself. While all experiments were distinct and independent, participants from one given year revealed a pattern consistent with other participants of the same year.

Fourth year participants displayed higher situational awareness and creativity. They walked more often around the room searching in a wider radius, focusing their attention at both close-range and long-range objects. In the latter case, they aimed the device at an object across the room but quickly realised that a closer inspection was required, similar to how human eyes behave. Should an object be in a high, out of reach place, such as in Figure 6.2, the participants then immediately searched for another. If a suitable object was not readily found or detected, a few participants tried to come up with their own solutions as shown in Figures 6.3a and 6.3b.

Third year participants walked less regularly and tended to focus on objects within grasp. In this year, one of the questions asked for a hexagon. However – and as hypothesised – no hexagons were found, since regular hexagonal shapes

Figure 6.2: Long range (detection-wise) objects



(a) Line parallelism question

(b) Angles question

Figure 6.3: Creative solutions figured out by fourth year participants

are rather uncommon among day-to-day objects. Still the participants were able to effortlessly and correctly describe what they were looking for. The main struggle lied in recalling the distinction between parallel and perpendicular lines. To deal with this, most participants simply employed trial and error methods, under the assumption that if a certain line arrangement was not parallel, it would be perpendicular and vice-versa. Although much less apparent, some creativity was also displayed (Figure 6.4).

Finally, second year participants barely walked around the room. Not only would they just focus on the objects atop the table (Figure 6.1), displaying limited awareness, but would also look through the device while slowly turning around, as if recording a video of the nearby objects. This highly contrasts with the spot-and-aim usage pattern of the third and fourth years, either limiting the amount

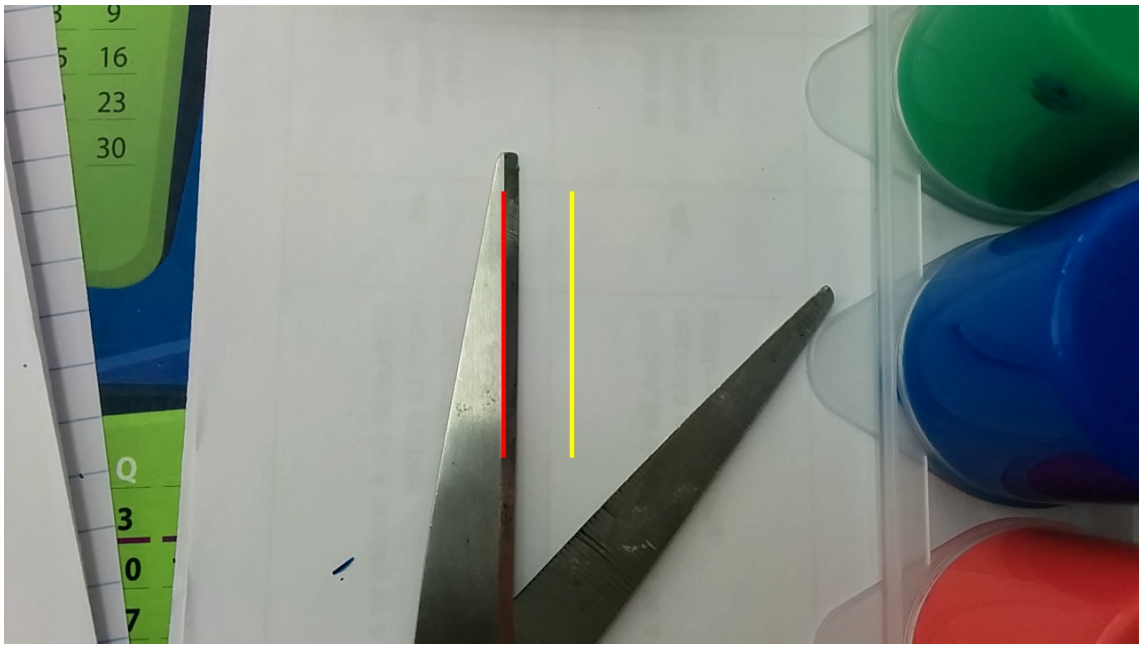


Figure 6.4: Another creative solution for a line parallelism question, which several third year participants struggled in recalling the difference

of objects searched or, in very few cases, accidentally detecting a fitting object and answering the question. As stated in Section 6.2, this year displayed some difficulty in reading the question text, sometimes taking as much as half of the time allotted. One of the questions asked for a pentagon, which was also nowhere to be found, and like the third year, another question asked for a hexagon. Unlike the third year, some participants had difficulties in recalling the shape, despite having learned it in class, confusing it with the pentagon. If the latter had been asked for earlier, most managed to recall it by process of elimination. As a side note, one particular participant benefited from the motion sensor based detection, for she was partially handicapped on her left hand and could only muster enough dexterity to hold the device, and would not be able to conduct the experiment if any sort of touch input was required. Considering the small radius of operation, most answers were consequently similar (Figure 6.5). Another pattern emerged from the answers themselves: most answers depict single, well defined, drawn shapes, whereas the other years' answers depicted compound and/or physical objects. Figures 6.6a through 6.6f display this difference.

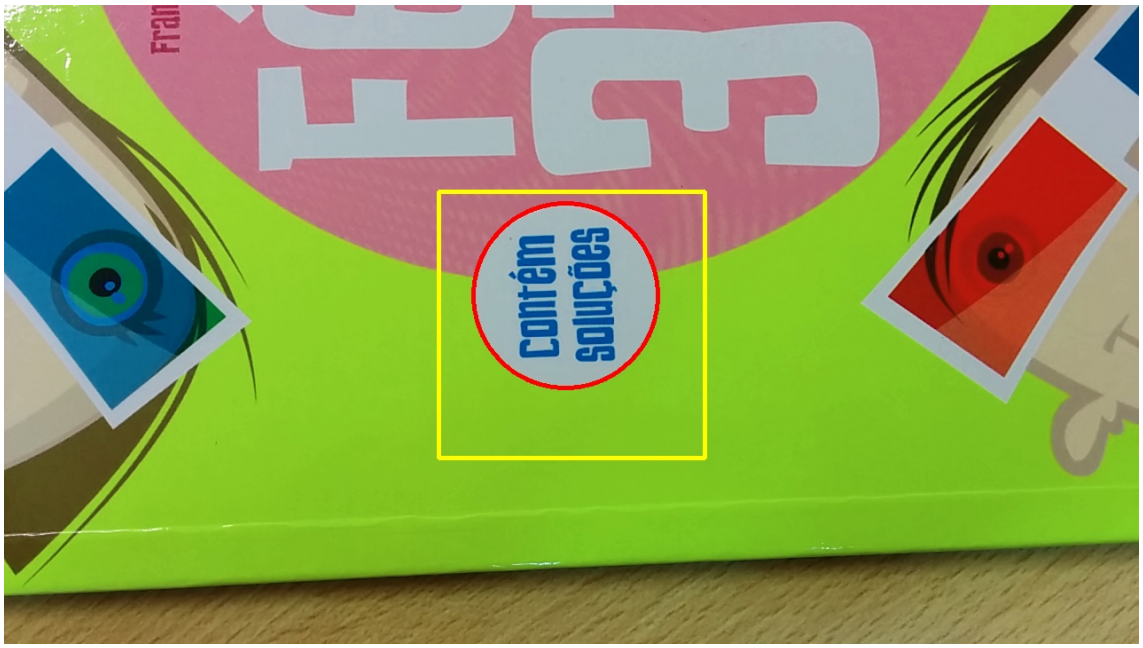
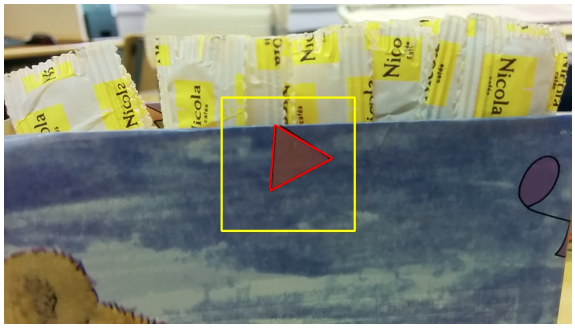
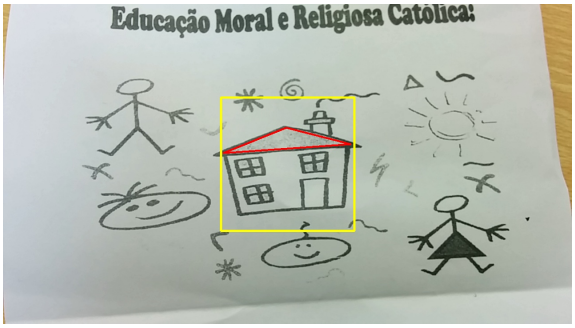


Figure 6.5: Second year's most answered shape

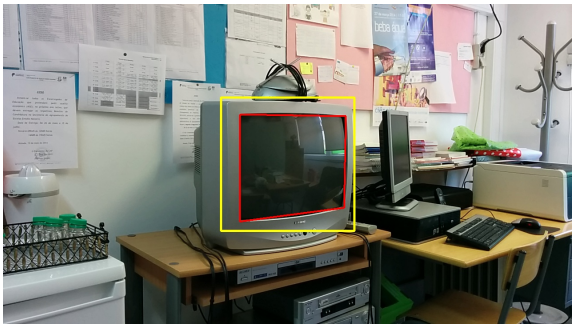


(a) Second year triangle answer

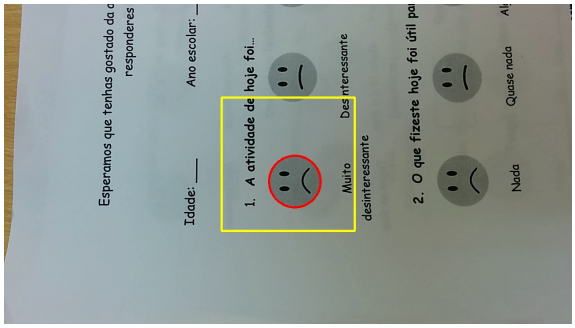


(b) Third year triangle answer

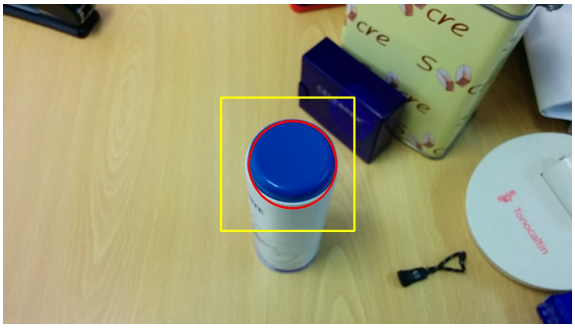
(c) Second year square answer



(d) Third year square answer



(e) Second year circle answer



(f) Fourth year circle answer

Figure 6.6: Answer comparison (for questions in common) between second year and other years

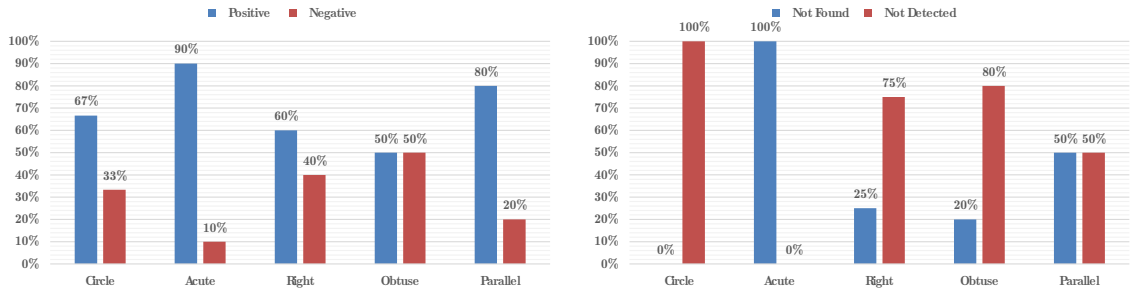
In the end, most participants were pleased with their performance, even in the cases where the device could not detect the object. Some participants asked for a second run to improve their and others for more challenging questions, while some wanted to know if someone had performed better, spurring the competitive nature of this age.

6.4 Result Analysis

This section is divided into two parts: one concerning user results and the another questionnaire results. The first part aims to shed some insight into how users performed per class, as a whole and, to some extent, the detection system's performance in a real scenario; the second to visualise user feedback. These results can be examined in greater detail in Appendix B.

6.4.1 User Results

Starting with the fourth year, Figure 6.7 reveals that participants were able to answer correctly most of this highest difficulty set of questions. Except for the obtuse angle question, all other questions had a higher positive answer ratio. Most of their negative results were due to the prototype not detecting the required shape or notion, as opposed to not finding it. Not found is differentiated from not detected by the number of detection attempts. If no attempts were made, it means the participant could not find the respective shape or notion in time. Although the motion based detection system employed can trigger accidental detections, most participants were constantly moving around the room and only stood still enough when willingly detecting an object.

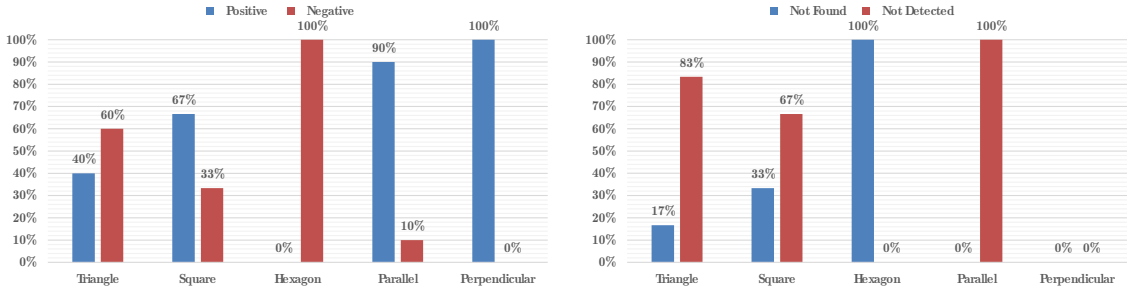


(a) Positive vs negative answers

(b) Not found vs not detected answers

Figure 6.7: Fourth year performance results

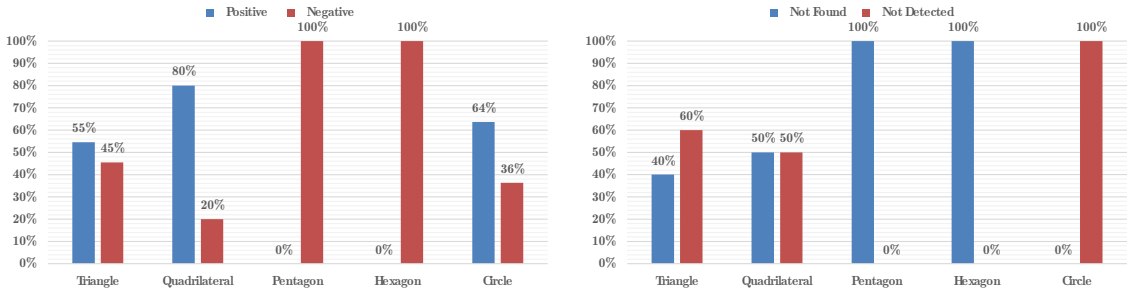
Third year participants also displayed a high amount of knowledge, answering most questions correctly as can be seen in Figure 6.8. Once again, most negative results were caused by lack of detection success from the prototype. As stated previously, no hexagon-shaped objects were found in the room, thus penalising the participants in when compared to the fourth year.



(a) Positive vs negative answers (b) Not found vs not detected answers

Figure 6.8: Third year performance results

Figure 6.9 shows that the second year participants were further penalised since pentagons also could not be found. Otherwise they managed to answer the rest of the questions correctly in most cases. At this level, the ratio between not found and not detected answers is more even. This could be attributed more to a slower thought process rather than actual lack of knowledge, considering these participants took significantly longer to read the questions presented, consequently restricting even further the alloted search time.

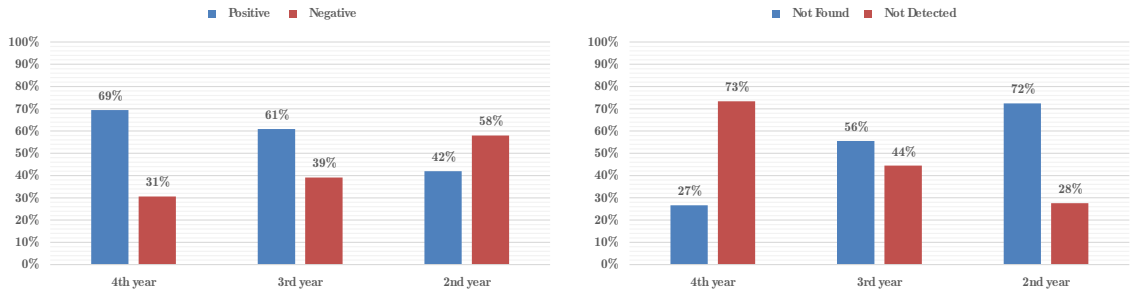


(a) Positive vs negative answers (b) Not found vs not detected answers

Figure 6.9: Second year performance results

It is relevant to visualise the big picture concerning these results, to better gauge the knowledge difference between years. Hence, Figure 6.10 displays the aggregated results from all years. The fourth year shows a higher discrepancy between both positive/negative answers, favouring positive answers, as well as not found/detected answers, towards a higher detection failure ratio. The third

year evens out the ratios and the second year inverts them. In this respect, and given the fact that two shapes were not available, all it is possible to remark is that most, if not all, participants indeed knew what to look for, even if the overall results point otherwise.

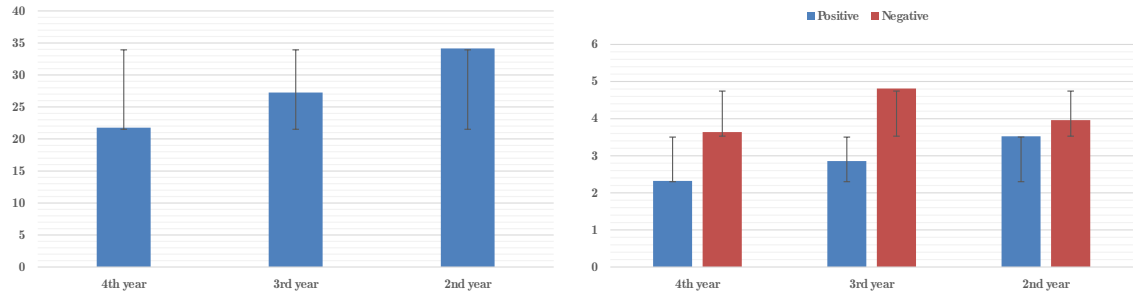


(a) Positive vs negative answers

(b) Not found vs not detected answers

Figure 6.10: Aggregate performance results

Regarding completion time, Figure 6.11 shows that, when all went well and the participants correctly completed a given question, they were quite fast. Fourth year participants took approximately a third of the alloted, with third year participants trailing right behind. Second year participants took sensibly ten seconds longer, but still finished comfortably with time to spare. The only way to fail a question was to exhaust the sixty seconds alloted time, thus it made no sense to add that information in Figure 6.11a. The number of detection attempts are quite low as well. Most of the cases, participants would only require a small distance adjustment in order to positively identify a geometric shape or notion. In the cases where detection ended up in failure most participants would usually switch between similar objects until time ran out, except second year participants. They would keep aiming at the same object hoping detection would succeed before time was up, which explains why their value is lower than third year's.



(a) Mean positive answer time (s) per question

(b) Mean number of detection attempts per question

Figure 6.11: Aggregate completion results

Within the completion time one can look at how much to a degree that time is inflated by observing the amount of time spent in detection. As can be seen in Figure 6.12, it is relatively low on positive answers, which signifies the geometric shapes or notions were being successfully detected rather quickly. On the other hand, detection hogged nearly half of the allotted time during negative answers.

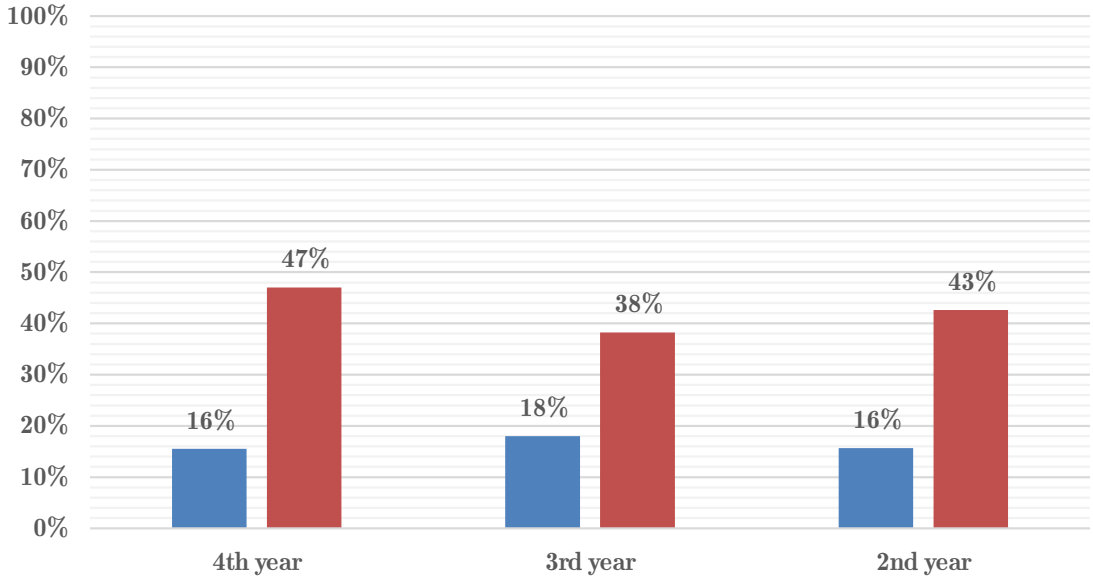


Figure 6.12: Mean time spent detecting per question

6.4.2 Questionnaire Results

The Likert-scale results from the questionnaire show that participants found the experiment useful to acquire or consolidate knowledge concerning geometry (Figure 6.13). Turning this prototype into a full-fledged game would capture the interest of the participants according to Figure 6.14, with one participant commenting that she never came across a mobile device game about geometry that utilized the device's camera in such manner. Opinions were somewhat more divided regarding the difficulty of the experiment as can be verified in Figure 6.15. Some participants felt it was too easy whereas others felt the time constraint was too severe. One participant commented that the motion sensor approach was not very suitable for her because her hands shake quite a bit by nature and therefore was finding rather challenging to trigger detection, which in turn increased the difficulty of the experiment. Figure 6.16 shows most participants are used to interacting with mobile devices on a regular basis.

Below are a few interesting remarks made by the participants. They are provided as written by the participants, therefore reader grammar discretion is advised. All questionnaire graphs and comments can be found in Appendix C.

> *"It was very fun and is a good way of learning and testing our brain"*

> *"I found it very easy, the game helps us learn more things about angles and geometric shapes"*

> *"I think this game is very interesting. It tests our mathematical ability as it amuses us"*

> *"I never saw a game this cute with a camera"*

> *"The game was fun, I managed to find almost everything and it went like this (triangle, square, circle, rectangle, pentagon). That is all for today, thank you João"*

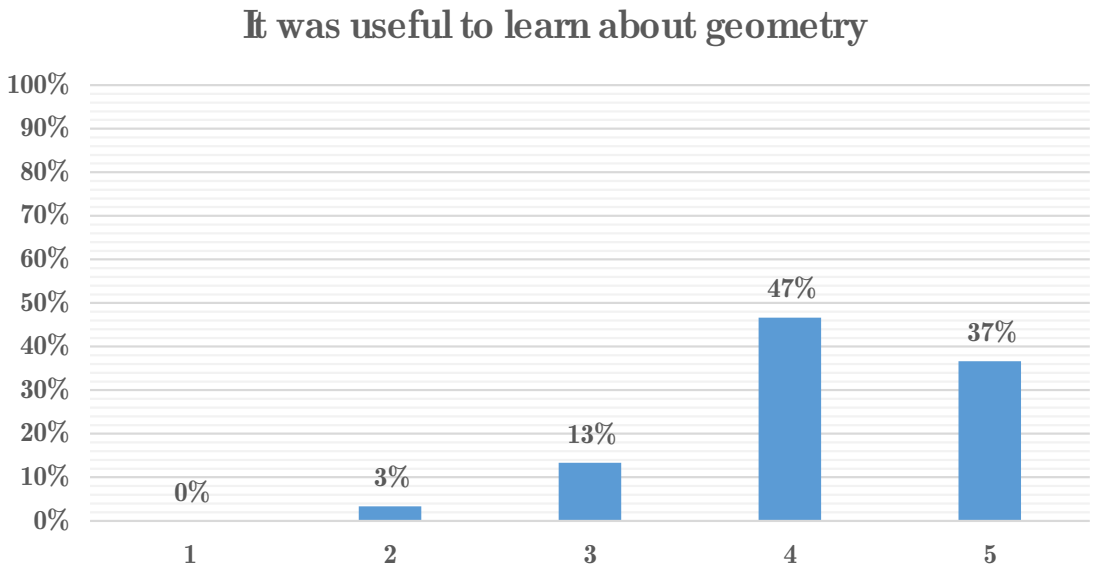


Figure 6.13: Usefulness histogram - 5 is best

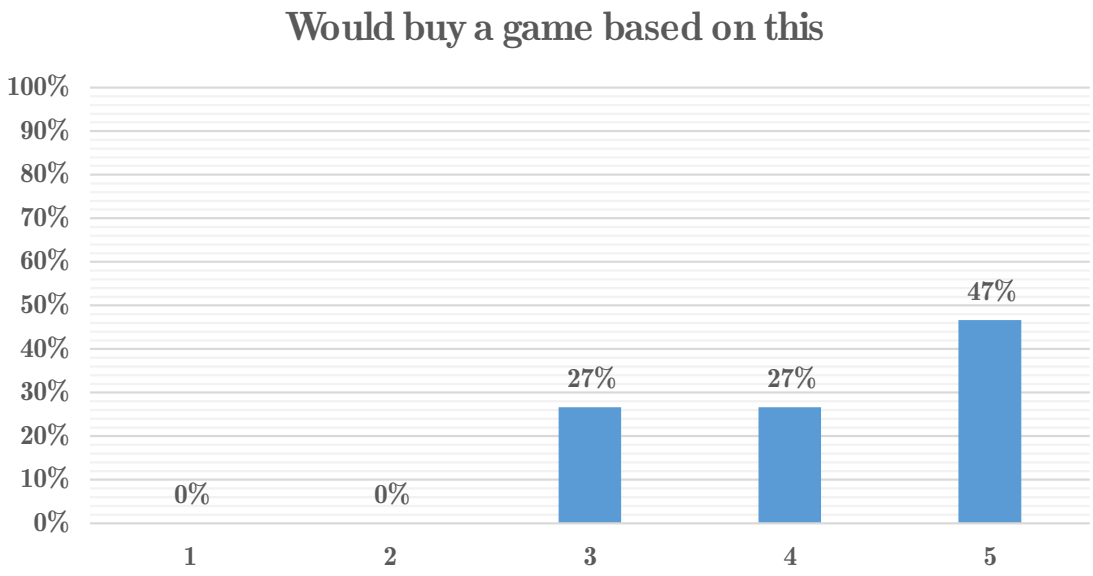


Figure 6.14: Potential as an educational game - 5 is best

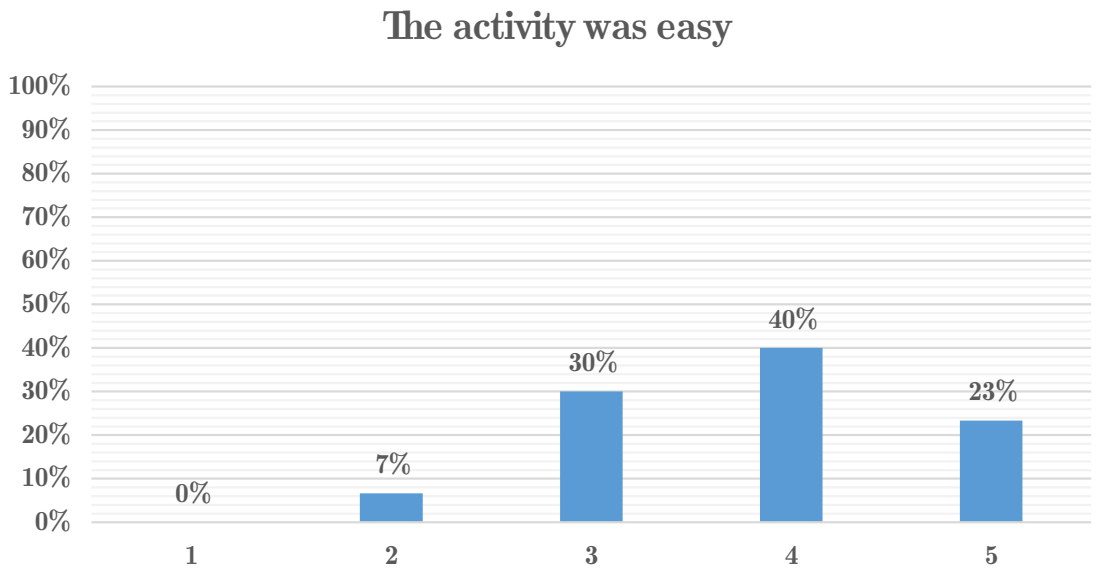


Figure 6.15: Difficulty histogram - 5 is best

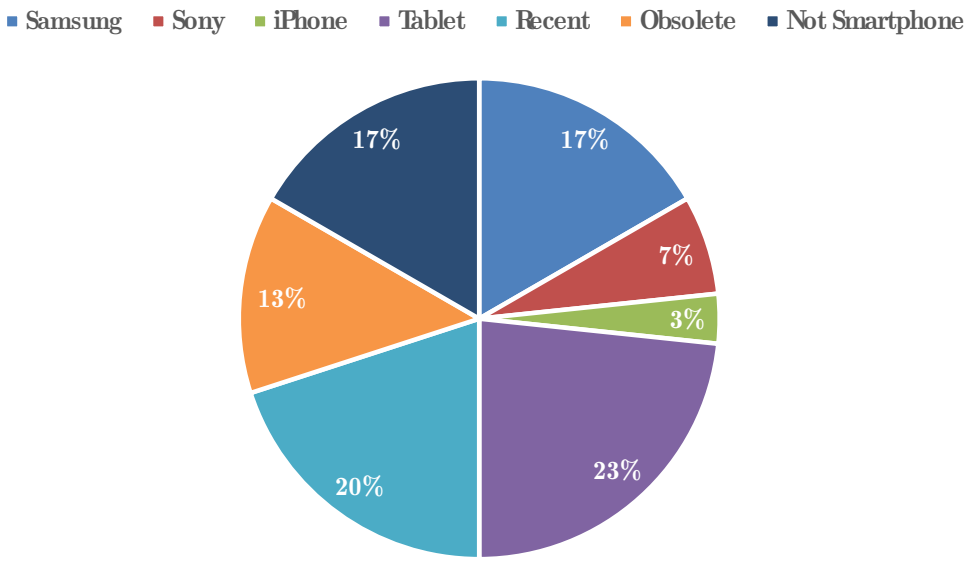


Figure 6.16: Mobile devices the participants use at home, but not necessarily own. Obsolete refers to devices three or more years old

6.4.3 Discussion

The experiment suggested most students in the 6 - 10 age range have a decent amount of geometric knowledge consolidated. Their high degree of adaptability to newer technologies allows for them to embrace more easily approaches to education based on such technologies. Although not the focus of this experiment, a few prototype related results could be included due to extra logging for completeness sake. Recall that an older, less optimised, version of the prototype was used at the time of the experiment, and as such these prototype results do not reflect the performance found on the tests in Chapter 5.

7

Conclusion and Future Work

The current chapter will conclude this dissertation in Section 7.1. Section 7.2 delves on what future work could be done to further improve the detection system.

7.1 Conclusion

This dissertation presented a real-time mobile detection system developed for several geometric shapes and notions. It is designed as a library specifically with mobile devices in mind, and its C++ implementation allows for both speed and portability, ergo it is not limited to one form factor or mobile operating system. Many geometric shapes can be detected in real-time by the library, including not only triangles, rectangles, squares, pentagons, hexagons and circles, but also geometric notions such as angles, line parallelism and symmetry. A wide range of customisation settings are available, from the ROI size to the edge extraction approach used.

For the prototype, an Android application integrating the library was developed for smartphones, which makes use of inherent features such as the accelerometer in order to effortlessly trigger detection. Evaluation tests have shown it to be efficient enough to meet the target frame rate during detection on medium-spec devices, a large sum of the devices in use nowadays. They have also shown resolution, in contrast to processing power or available memory, has the highest

impact on performance, although it remained acceptable at the elevated FullHD (1920x1080) resolution. Since mobile devices tend to include cameras with ever increasing capture and recording resolutions, it will certainly pose a higher issue down the line as processor upgrades are much more conservative and therefore cannot keep up. Certain hardware features like Optical Image Stabilisation help in improving performance. By mitigating the amount of blur caused by involuntary movements, it allows for sharper images with clearly defined features which can be processed faster and lead to more accurate detections.

To assess the educational factor of the application, an user study was conducted involving elementary school students. They were asked to search for several objects resembling geometric shapes or notions and use the application to detect the respective shape or notion. Students showed to embrace this approach, describing it as challenging yet fun. It prompted them to recall geometry concepts they learned in class, helping consolidate that knowledge. They also outlined the aspect of being able to use the application at home and explore at their own pace. Although an older prototype was used, it performed acceptably, recognizing geometric shapes and notions in a myriad of objects.

Overall, all the proposed goals were achieved and the library can be deemed ready to be deployed on a mobile game. While one could argue the detection system to be overly simplistic in its conception when comparing to available desktop-oriented solutions for geometry detection, such as the one tested during the exploratory stage, the author firmly believes employing a simple method far outshines using a more complex approach to achieve the same objective. In the end the remaining feeling is that this system contributes to expanding mobile real-time detection to an area that was lacking more viable options.

7.2 Future Work

There is always room for improvement. A few key areas could be improved in future developments:

- Expand the array of geometric shapes and notions detected by the library
- Allow the library to detect different shapes and notions simultaneously
- Make the library settings customisable dynamically
- Adjust the prototype for accommodating different detection triggers

To better measure geometric knowledge consolidation by children, further user studies are warranted. An experiment involving before and after test taking would help to understand how using the application incurred in knowledge acquisition or recollection. Also more feedback could be obtained from teachers regarding to how this system could be used in class scenarios to help maximize educational gains.

Bibliography

Quentin Bonnard (2012). *Augmented Paper for Learning Geometry*. Last access: February 2014. URL: http://craft.epfl.ch/lang/en/PaperTangibleInterface.

Quentin Bonnard and Himanshu Verma (2012). "Paper interfaces for learning geometry". In: *...Learning for 21st Century ...* DOI: 10.1007/978-3-642-33263-0_4. URL: http://link.springer.com/chapter/10.1007/978-3-642-33263-0_4.

John Canny (Nov. 1986). "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6, pp. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4767851.

Wenqing Chen et al. (2010). "A Fast Geometry Figure Recognition Algorithm Based on Edge Pixel Point Eigenvalues". In: August, pp. 297–300. URL: https://academypublisher.com/~academz3/proc/iscsct10/papers/iscsct10p297.pdf.

CNET (2013). *Android dominates 81 percent of world smartphone market*. Last access: January 2014. URL: http://news.cnet.com/8301-1035_3-57612057-94/android-dominates-81-percent-of-world-smartphone-market/.

H. David Douglas and K. Thomas Peucker (1973). "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10 (2), pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727.

Richard O. Duda and Peter E. Hart (Jan. 1972). "Use of the Hough Transformation to Detect Lines and Curves in Pictures". In: *Commun. ACM* 15.1, pp. 11–15.

ISSN: 0001-0782. DOI: 10.1145/361237.361242. URL: http://doi.acm.org/10.1145/361237.361242.

Robert Fisher et al. (2003). *Adaptive Thresholding*. Last access: April 2014. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm.

Google (2013). *Android KitKat*. Last access: February 2014. URL: http://developer.android.com/about/versions/kitkat.html.

C. Harris and M. Stephens (1988). "A Combined Corner and Edge Detector". In: *Procedings of the Alvey Vision Conference 1988*, pp. 23.1–23.6. DOI: 10.5244/C.2.23. URL: http://www.bmva.org/bmvc/1988/avc-88-023.html.

Ilina Kareva (2011a). *Simple shape detector – Android Application*. Last access: January 2014. URL: http://ilinakareva.wordpress.com/2011/05/08/simple-shape-detector-android-application/.

— (2011b). *Testing my thesis apps in a local primary school*. Last access: January 2014. URL: http://ilinakareva.wordpress.com/2011/05/10/testing-my-thesis-apps-in-a-local-primary-school/.

H Kaufmann (2004). "Geometry education with augmented reality". In: *Vienna University of Technology* 9225889.

Hannes Kaufmann and Andreas Dünser (2007). "Summary of Usability Evaluations of an Educational Augmented Reality Application". In: *Proceedings of the 2Nd International Conference on Virtual Reality*. ICVR'07. Beijing, China: Springer-Verlag, pp. 660–669. ISBN: 978-3-540-73334-8. URL: http://dl.acm.org/citation.cfm?id=1770090.1770165.

Hannes Kaufmann and Dieter Schmalstieg (2002). "Mathematics and geometry education with collaborative augmented reality". In: *ACM SIGGRAPH 2002 conference abstracts and applications on - SIGGRAPH '02*, p. 37. DOI: 10.1145/1242073.1242086. URL: http://portal.acm.org/citation.cfm?doid=1242073.1242086.

R Laganière (2011). *OpenCV 2 computer vision application programming cookbook*. ISBN: 9781849513241. URL: http://books.google.com/books?hl=en\&lr=\&id=OC7jc8zWjlkC\&oi=fnd\&pg=PT7\&dq=OpenCV+2+computer+vision+application+programming+cookbook\&ots=CcttIXnPye\&sig=RFqo40kBFgyvE1JvAjxXUqBmihw.

ABC News (2011). *To a Baby, a Magazine Is 'an iPad That Does Not Work'*. Last access: January 2014. URL: http://abcnews.go.com/blogs/technology/2011/10/to-a-baby-a-magazine-is-an-ipad-that-does-not-work/.

Kimberley Osberg (1997). "Spatial Cognition in the Virtual Environment". In: *Technical R-97-18. Seattle: Human Interface Technology Lab*. URL: http://www.hitl.washington.edu/projects/education/puzzle/spatial-cognition.html.

N Otsu (1979). "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, pp. 62–66. ISSN: 0018-9472. DOI: 10.1109/TSMC.1979.4310076. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4310076.

Marc Pomplun (2013). *Compactness*. Last access: January 2014. URL: http://www.cs.umb.edu/~marc/cs675/cvs09-12.pdf.

E Rosten and T Drummond (2006). "Machine learning for high-speed corner detection". In: *Computer Vision–ECCV 2006*. Lecture Notes in Computer Science 3951. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz, pp. 1–14. DOI: 10.1007/11744023_34. URL: http://link.springer.com/chapter/10.1007/11744023_34.

Richard Szeliski (2011). *Computer Vision*. Texts in Computer Science. London: Springer London. ISBN: 978-1-84882-934-3. DOI: 10.1007/978-1-84882-935-0. URL: http://link.springer.com/10.1007/978-1-84882-935-0.

The Telegraph (2013). *Smartphone sales account for more than half of global mobile phone market*. Last access: January 2014. URL: http://www.telegraph.co.uk/technology/mobile-phones/10448819/Smartphone-sales-account-for-more-than-half-of-global-mobile-phone-market.html.

Mohd Firdaus Zakaria, Hoo Seng Choon, and Shahrel Azmin Suandi (2012). "Object Shape Recognition in Image for Machine Vision Application". In: *International Journal of Computer Theory and Engineering* 4.1, pp. 76–80. ISSN: 17938201. DOI: 10.7763/IJCTE.2012.V4.428. URL: http://www.ijcte.org/show-40-424-1.html.

Paper Testbed

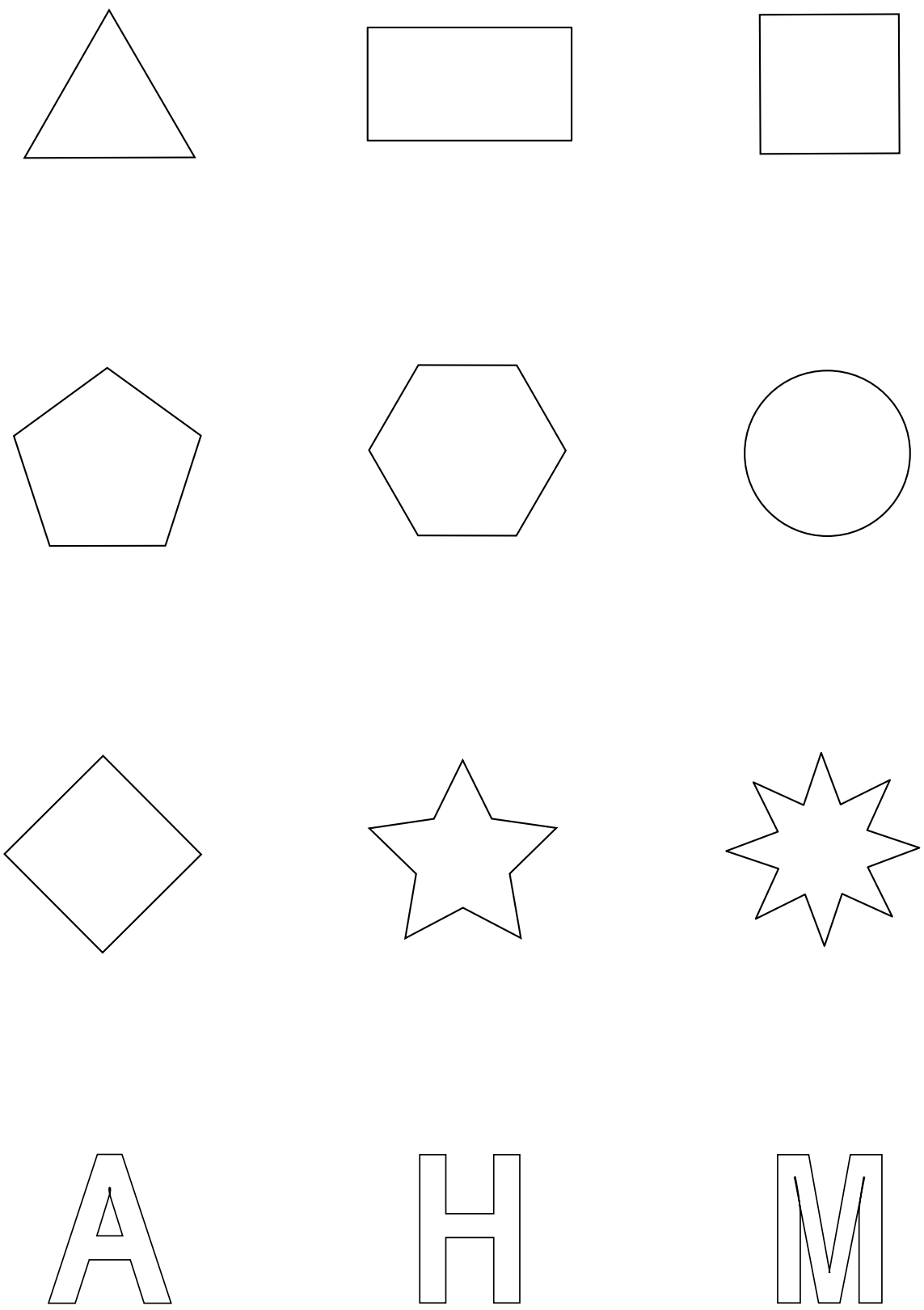


Figure A.1: The “Paper” testbed

B

User Results

Questionnaire

Esperamos que tenhas gostado da atividade de hoje, que termina assim que responderes a estas perguntas

Idade: ___ Ano escolar: ___

1. **A atividade de hoje foi…**

Muito desinteressante

Desinteressante

Normal

Interessante

Muito interessante

2. **O que fizeste hoje foi útil para aprender…**

Nada

Quase nada

Algumas coisas

Muitas coisas

Quase tudo sobre formas geométricas

3. **Pedias aos teus pais para comprar este jogo?**

Nunca (não vejo o interesse)

Provavelmente não

Talvez (se me lembrasse)

Provavelmente sim

Claro que sim

4. **O jogo foi…**

Muito difícil

Difícil

Normal

Fácil

Muito fácil

5. Gostavas de voltar a jogar este jogo?

Nunca	Poucas vezes	Talvez	Algumas vezes	Muitas vezes

6. Quantos dias por semana jogas no telemóvel ou tablet?

Nunca	Pouco (1 a 2 dias por semana)	Alguns (3 a 4 dias por semana)	Muito (5 a 6 dias por semana)	Todos os dias

7. O telemóvel onde costumas jogar é teu?

Sim	
Não	

8. Qual é o modelo do telemóvel ou tablet onde costumas jogar? (por exemplo Galaxy S)

_____

O que pensas é muito importante para nós, por isso se tiveres comentários ou opiniões em relação ao jogo que nos estás a ajudar a construir, escreve aqui:

We hope you enjoyed today's activity, which ends as soon as you finish these questions

Age: ___ Year: ___

1. Today's activity was...

Very uninteresting

uninteresting

Normal

Interesting

Very interesting

2. What you did today was useful to learn...

Nothing

Almost nothing

Some things

Many things

Almost everything about geometrical shapes

3. Would you ask your parents to buy you this game?

Never (I don´t see the point)

Probably not

Maybe (if I remembered)

Probably yes

Of course

4. The game was...

Very hard

Hard

Normal

easy

Very easy

5. Would you like to play this game again?

Never | A few times | Maybe | Sometimes | Many times

6. How many days a week do you play in a smartphone or tablet?

Never | A few (1 to 2 days a week) | Some (3 to 4 days a week) | Many (5 to 6 days a week) | Every day

7. Is the smartphone you use to play yours?

Yes	
No	

8. What model is the smartphone or tablet you use to play? (for example Galaxy S)

_____

What you think is very important to us, so if you have any comments or opinions regarding the game you are helping us build, write them here:

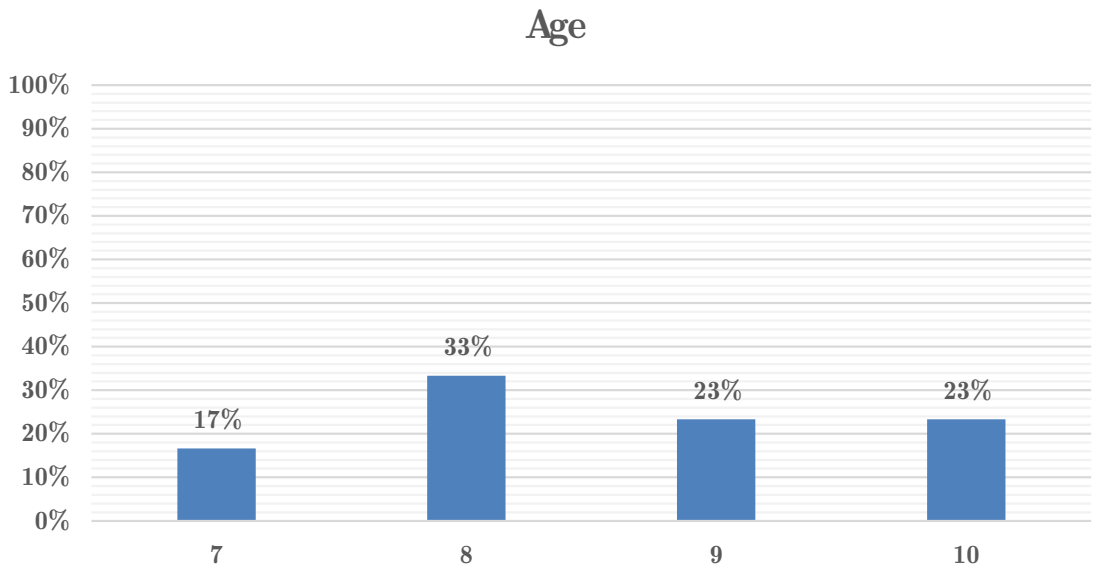


Figure C.1: Age

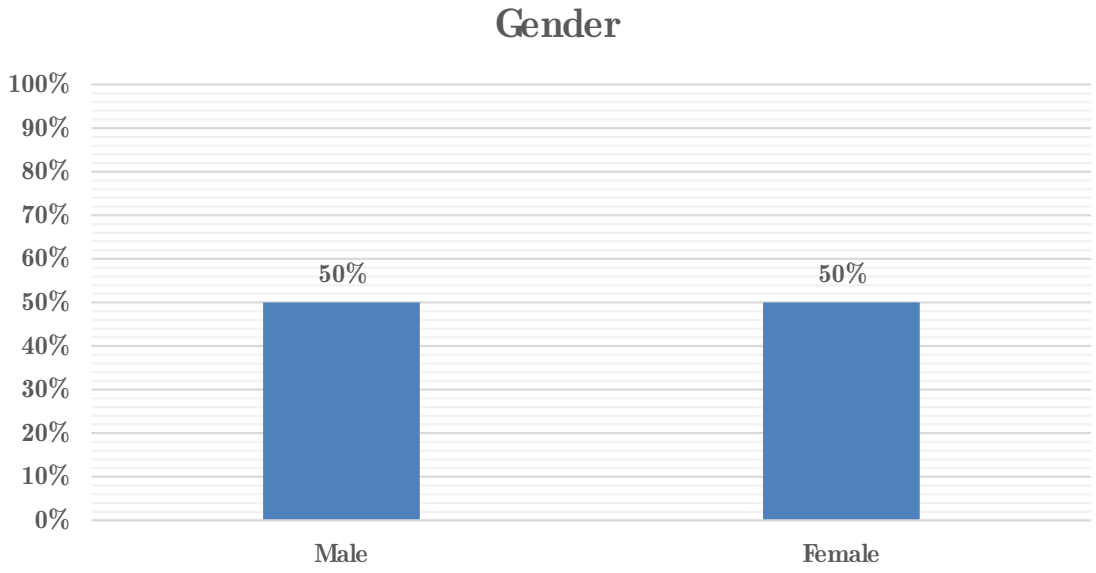


Figure C.2: Gender

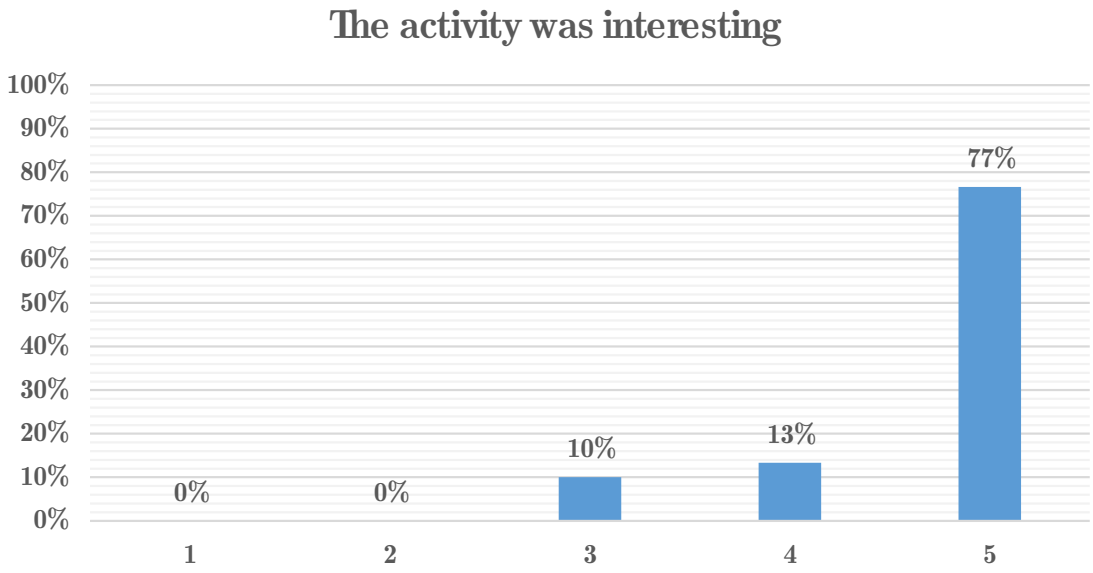


Figure C.3: Question 1

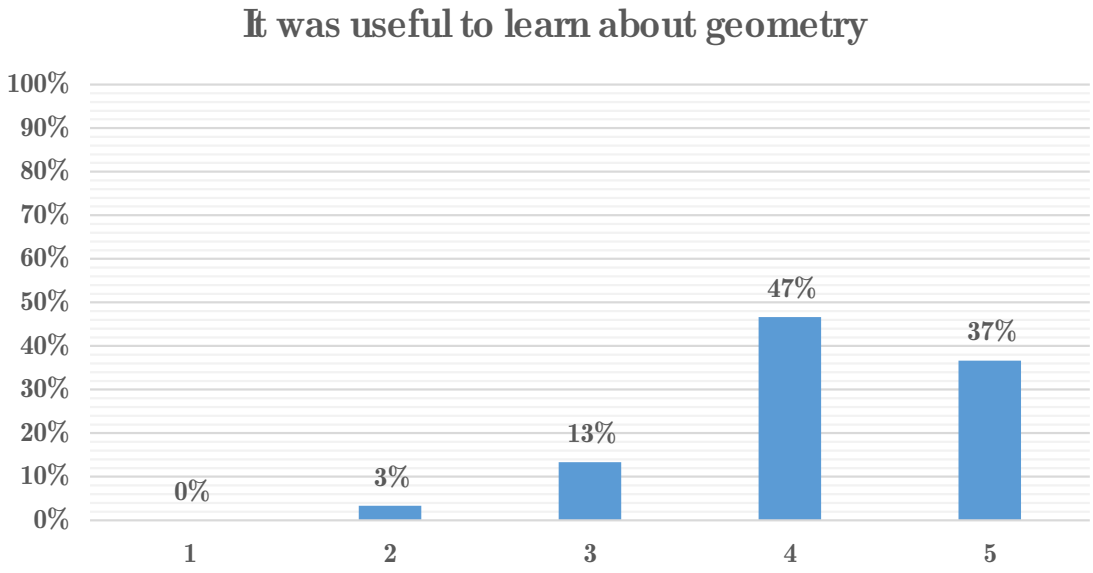


Figure C.4: Question 2

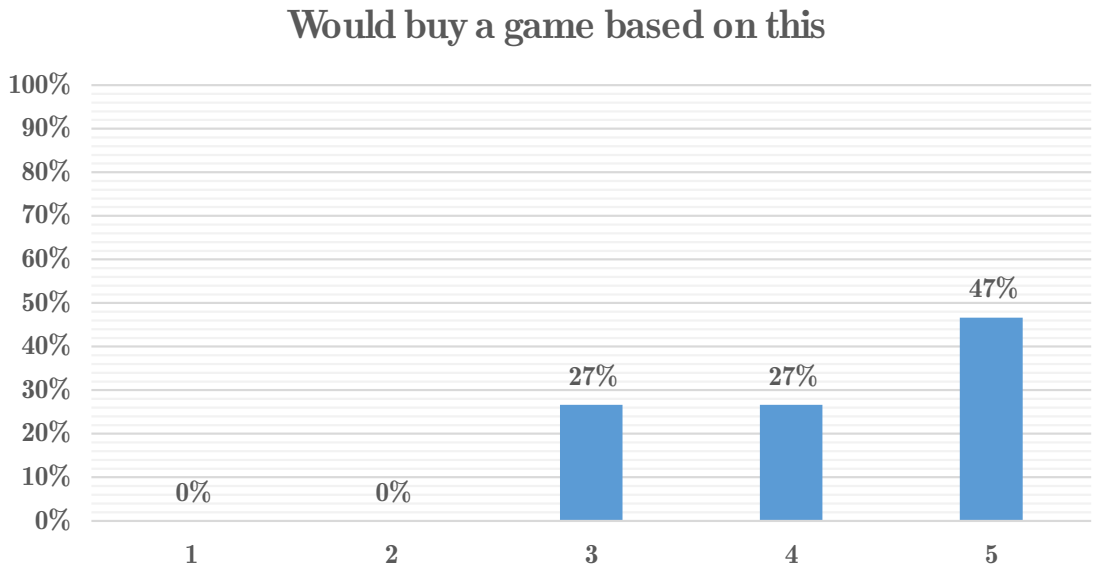


Figure C.5: Question 3

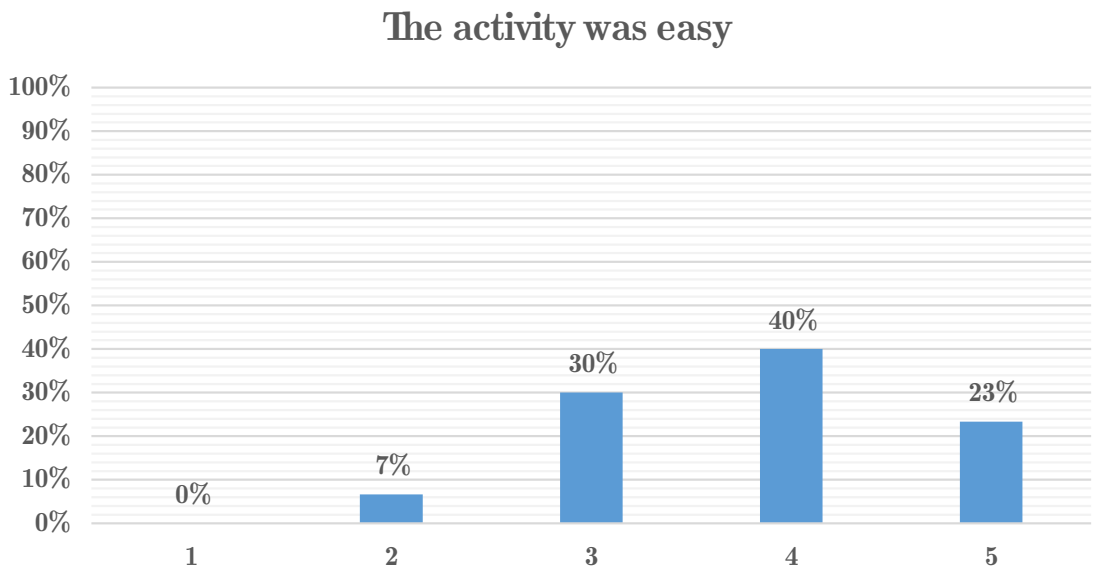


Figure C.6: Question 4

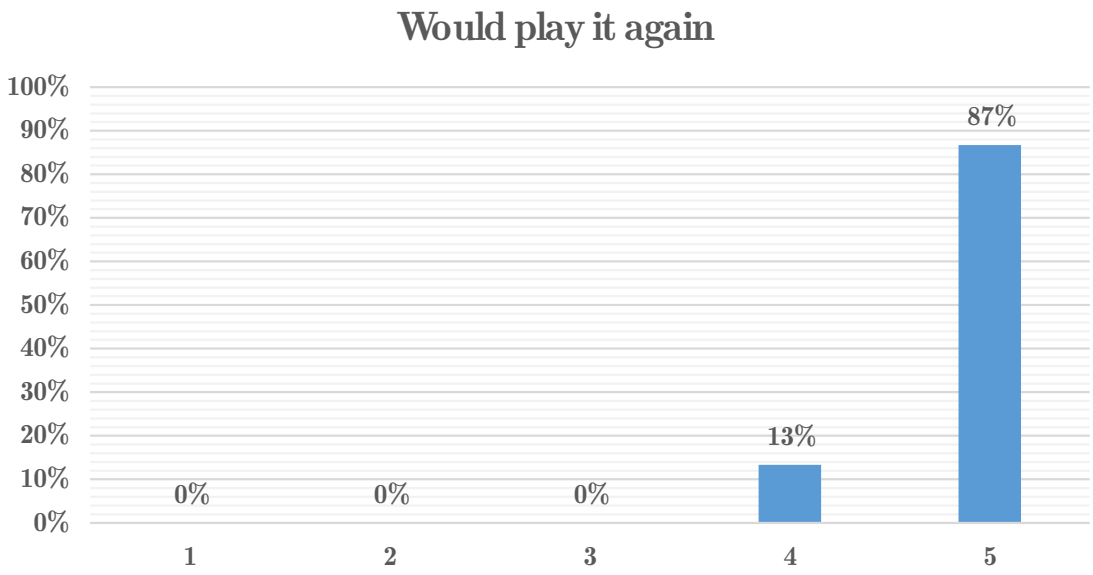


Figure C.7: Question 5

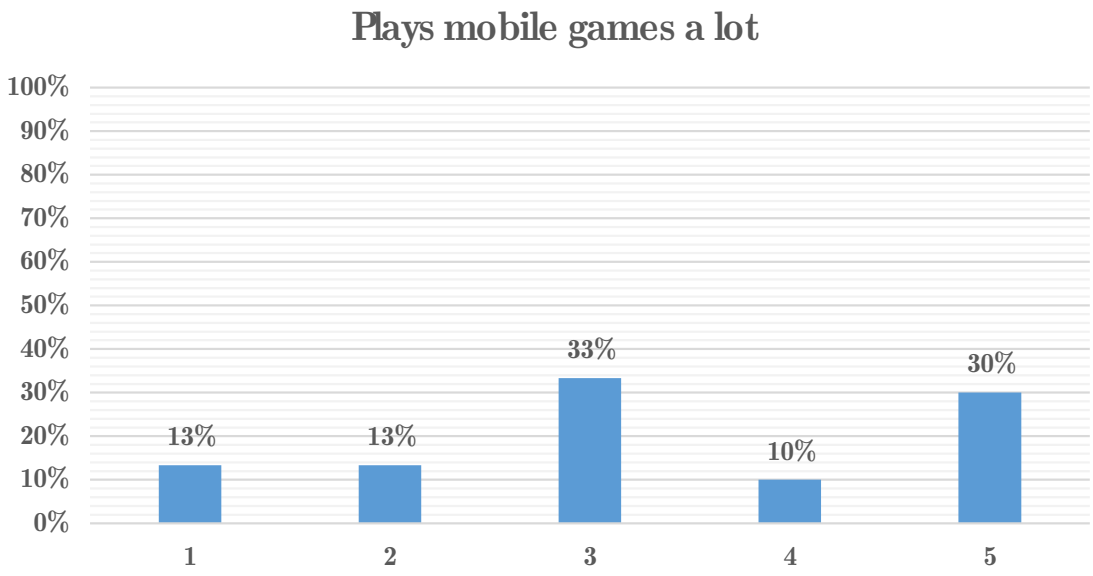


Figure C.8: Question 6

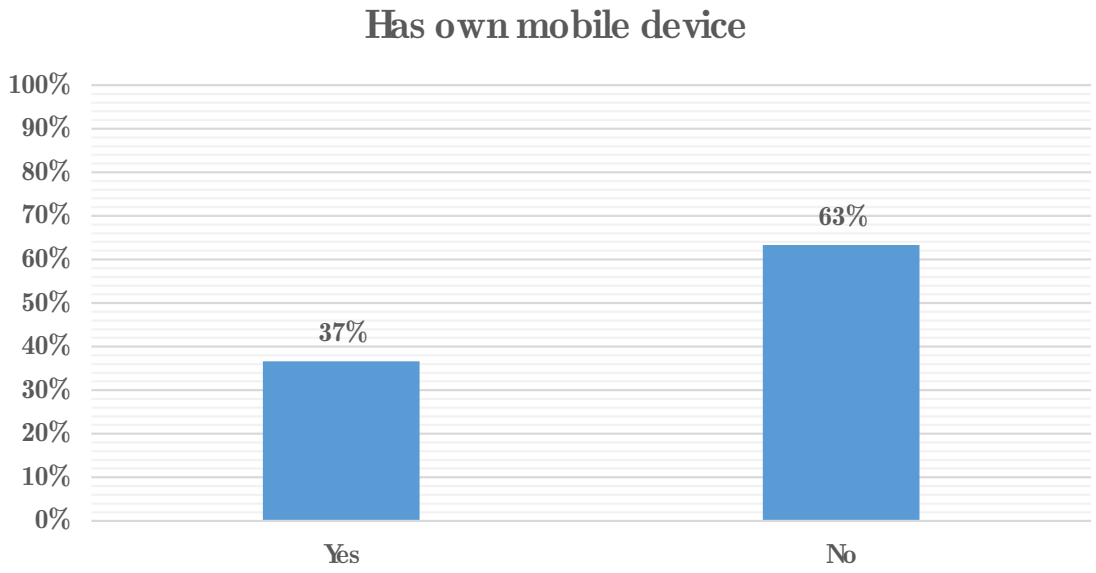


Figure C.9: Question 7

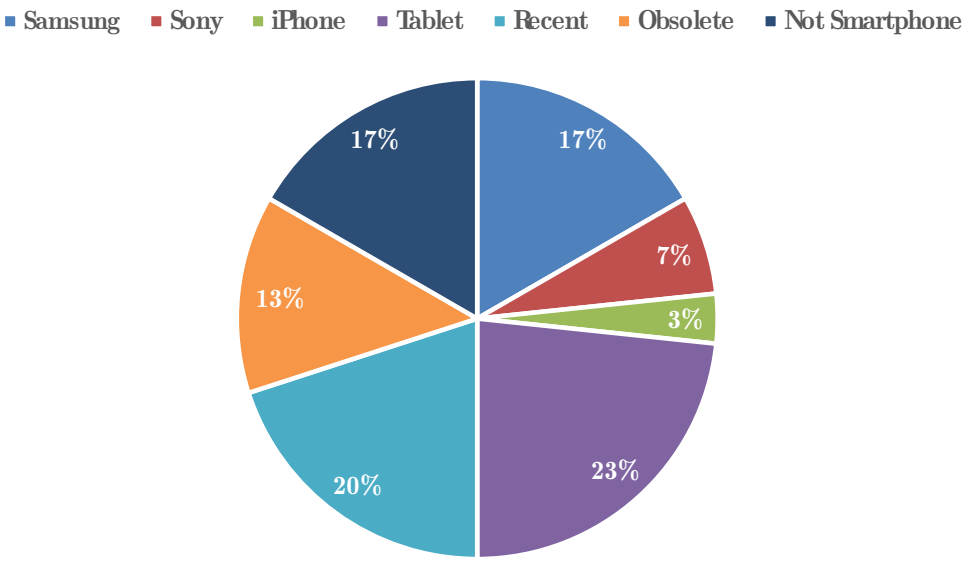


Figure C.10: Question 8

"It was very fun and is a good way of learning and testing our brain"

"I liked the game very much, it is very interesting, I learned many things about geometry and the teacher's room"

"Very good"

"I think it was very fun and it should be a little more difficult"

"To me this game was good but a bit difficult"

"I found it very easy, the game helps us learn more things about angles and geometric shapes"

"In my opinion one should touch and the phone detect"

"I liked this game very much and think I shook a little"

"I found it interesting, fun"

"I liked this game very much"

"I think it was an interesting game for learning. Hope to play again"

"I liked it"

"It was very good, but I think it would be even better if it had more questions"

"I think this game is very interesting. It tests our mathematical ability as it amuses us"

"I never saw a camera this cool"

"I liked it very much and it should not change anything"

"I never saw a game this cute with a camera"

"(Questions should have) three minutes or four"

"I liked it very much, would like to stay longer and would play it every day"

"I enjoyed playing, it was fun and cute"

"I needed more time to explore but the game was interesting"

"This class was fun"

"I found the game very fun, loved playing but it was a little difficult, I couldn't find the objects to play with"

"The game was fun, I managed to find almost everything and it went like this (triangle, square, circle, rectangle, pentagon). That is all for today, thank you João" "I like this game very much, it was very cool, fantastic"

"I really liked it"